PATENT ABSTRACTS OF JAPAN

(11)Publication number : 06-068053
(43)Date of publication of application : 11.03.1994

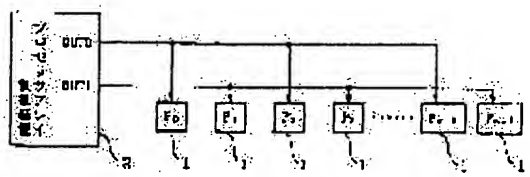
(51)Int.Cl. G06F 15/16

(21)Application number : 04-221592
(22)Date of filing : 20.08.1992

(71)Applicant : TOSHIBA CORP
(72)Inventor : FUJII HIROSHIGE
YOSHIDA TAKASHI
SATOU HISATOMO
TAKAHASHI MASASHI

(54) PARALLEL COMPUTER

(57)Abstract:
PURPOSE: To shorten the execution time and to effectively use processors even in the case of the degree of parallelism lower than the number of processors.
CONSTITUTION: A processor array consisting of n processors 1 and a processor array controller 2 which controls the processor array and extracts two kinds of instruction strings, which can be executed in parallel, from instruction strings to supply them to the processor array are provided, and one of two kinds of instruction strings supplied from output terminals OUT0 and OUT1 of the processor array controller 2 is received by each processor 1, and two kinds of instruction strings are executed in parallel by n processors.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平6－68053

(43)公開日　平成６年(1994)３月11日

(51)Int.Cl.[5]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　15/16	３９０　Ｔ	9190－5Ｌ		

審査請求　未請求　請求項の数５(全 27 頁)

(21)出願番号　特願平4－221592

(22)出願日　平成４年(1992)８月20日

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地

(72)発明者　藤井　洋重
神奈川県川崎市幸区小向東芝町１　株式会社東芝総合研究所内

(72)発明者　吉田　尊
神奈川県川崎市幸区小向東芝町１　株式会社東芝総合研究所内

(72)発明者　佐藤　寿倫
神奈川県川崎市幸区小向東芝町１　株式会社東芝総合研究所内

(74)代理人　弁理士　三好　秀和　(外１名)

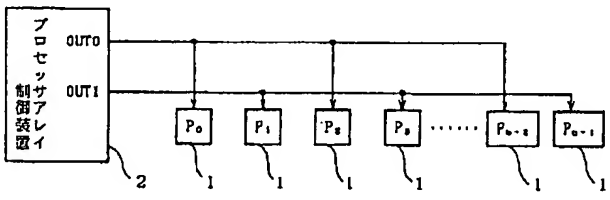
最終頁に続く

(54)【発明の名称】　並列計算機

(57)【要約】

【構成】　ｎ個のプロセッサ１からなるプロセッサアレイと、このプロセッサアレイを制御すると共に、命令列から並列実行可能な２種類の命令列を抽出してプロセッサアレイに供給するプロセッサアレイ制御装置２とを備え、プロセッサアレイ制御装置２の出力端子ＯＵＴ０，１から供給される２種類の命令列のうちの１種類を各プロセッサ１が受け取り、２種類の命令列をｎ個のプロセッサ１で並列実行する。

【効果】　並列度がプロセッサ数より小さい場合でも、実行時間の短縮とプロセッサの有効利用を図れる。



【特許請求の範囲】
【請求項１】　複数個のプロセッサからなるプロセッサアレイと、
このプロセッサアレイを制御すると共に、並列実行可能な複数種類の命令列を前記プロセッサアレイに供給するプロセッサアレイ制御手段とを備え、
前記プロセッサアレイ制御手段から供給される複数種類の命令列の中の１種類ずつを各プロセッサが受け取り、前記複数種類の命令列を前記複数個のプロセッサで並列実行することを特徴とする並列計算機。
【請求項２】　単一の制御プロセッサと、
該制御プロセッサからアクセス可能な第１のメモリと、
制御プロセッサから転送される命令情報にしたがって、処理を行う複数個の演算プロセッサと、
該演算プロセッサからアクセス可能な第２のメモリとから構成され、
さらに前記演算プロセッサには、
演算プロセッサでの実行を制御する命令情報の格納手段と、
該格納手段に、前記制御プロセッサからの命令情報を入力するか、第２のメモリに格納されている命令情報を格納するかを選択する手段と、
前記格納手段に格納されている命令情報にしたがって演算を実行する手段とを備えることを特徴とする並列計算機。
【請求項３】　個々の通信経路で接続されている複数の演算要素と、
解く問題のサイズに応じて演算に使用する演算要素と使用しない演算要素とを識別する識別手段と、
この識別手段によって識別された、演算に使用しない演算要素をバイパスするバイパス手段とを備えることを特徴とする並列計算機。
【請求項４】　個々の通信経路で接続されている複数の演算要素と、
各々の演算要素と個々に結合しているローカルメモリと、
解く問題のサイズに応じて演算に使用する演算要素と使用しない演算要素とを識別する識別手段と、
この識別手段によって識別された、演算に使用しない演算要素と結合しているローカルメモリを、演算に使用する演算要素から直接アクセス可能にするアクセス手段とを備えることを特徴とする並列計算機。
【請求項５】　複数のプロセッサがＮ次元ネットワークで相互接続され、
各プロセッサが、
隣接するプロセッサからメッセージを受信し、出力先を決定する出力先決定手段と、
出力先ごとに設けられ、前記出力先決定手段によって出力先が決定されたメッセージを受信した順に蓄える蓄積手段と、
蓄積されたメッセージを、決定された出力先のプロセッサに出力する出力手段とを備えたことを特徴とする並列計算機。
【発明の詳細な説明】
【０００１】
【産業上の利用分野】本発明は、並列計算機に関する。
【０００２】
【従来の技術】大規模な科学技術計算を高速に実行するために、並列計算機が期待されている。並列計算機の１つとしてＳＩＭＤ型並列計算機がある。ＳＩＭＤ型並列計算機は、複数のプロセッサ上に配置された複数のデータに対して、すべてのプロセッサで同じ命令を実行する。この場合、プロセッサ数分だけのデータに対する処理を並列に実行することになる。
【０００３】一般に、並列計算機上で実行しようとするプログラムには、並列度というものがあり、それ以上のプロセッサがあっても、速度は向上しない。例えば、繰り返し処理を並列実行するときに、１回の繰り返しを１台のプロセッサで実行することを考えると、１０００回の繰り返し処理を１００００台のプロセッサで実行しても、１０００倍以上には速度向上は望めない。したがって、プロセッサの数よりも少ない並列度の問題を解く場合には、実質的に使用されないプロセッサが多くなり、無駄である。
【０００４】問題の並列度が少ない場合に、実行時間の短縮とプロセッサの有効利用を考える必要がある。例えば、上述の例では繰り返し処理が複数命令からなり、その複数命令の中で並列に実行できる命令があれば、その命令を複数のプロセッサで並列実行できる可能性がある。しかしながら、このような複数命令の並列実行は、単一命令を全プロセッサが実行するというＳＩＭＤ型並列計算機では、実現するのは困難である。
【０００５】ここで、一般的な並列計算機について図６を用いて説明する。
【０００６】同図で、この並列計算機は、大きく分けて、１台の制御プロセッサ（ＣＰ）１１と、複数台の演算プロセッサ（ＰＥ）１２から構成される。制御プロセッサ１１には全体メモリ（Ｍ）１３が接続され、演算プロセッサ１２にはそれぞれローカルメモリ（ＬＭ）１４が接続される。制御プロセッサ１１と各演算プロセッサ１２は、グローバルバス１５で接続される。演算プロセッサ１２は、ネットワーク１６で相互接続される。ネットワーク１６の形態としては、格子結合、２進ｎキューブ結合、木結合等、多くのものが知られている。
【０００７】制御プロセッサ１１は、それ自体が一つの情報処理装置であり、全体メモリ１３に格納されているプログラム（機械語命令列）を読み込み、解読する。解読した命令が制御プロセッサ１１で実行すべき命令であれば制御プロセッサ１１自身で実行する。解読した命令が演算プロセッサ１２で実行すべき命令であれば、制御

プロセッサ１１はこの命令をグローバルバス１５を介して全演算プロセッサ１２に放送し、演算プロセッサ１２でこの命令の実行が行われる。

【０００８】制御プロセッサ１１は、グローバルバス１５を介して、すべての、あるいは特定の演算プロセッサ１２から、データの読み出し、および書き込みを行うことができる。また、同期バス１７を介して、すべての演算プロセッサ１２の状態の論理和を認識することができる。

【０００９】さらに、制御プロセッサ１１は、フロントエンド計算機１８と結合されている。フロントエンド計算機１８では、制御プロセッサ１１および演算プロセッサ１２で実行するプログラムの開発を行ったり、処理すべきデータの入出力を行う。

【００１０】演算プロセッサ１２は、制御プロセッサ１１から放送される命令に従って、自ローカルメモリ１４のデータに対して演算を行ったり、ネットワーク１６を介して他の演算プロセッサ１２とデータ通信を行ったりして処理を行う。

【００１１】演算プロセッサ１２は、例えば、図３７に示すような構成になっている。

【００１２】命令レジスタ２４には、制御プロセッサ１１から放送される命令が、１クロックごとに格納される。演算プロセッサ１２を構成する他のすべての要素、すなわち、アドレスレジスタ（ＡＲ）２６、データレジスタ（ＤＲ）２７、状態制御レジスタ（ＰＳＷ）３１、プロセッサ番号レジスタ（ＰＮＲ）３２、ＡＬＵ２９、レジスタファイル３０、通信制御部３３は、命令レジスタ２４の内容によって制御される。

【００１３】制御プロセッサ１１からは、一時期には常に同じ命令が全演算プロセッサ１２に対して放送され、全演算プロセッサ１２が一斉に同じ処理を行うことになる。いわゆるＳＩＭＤ（Single Instruction Multiple Data Stream）方式と呼ばれるものである。一方、各演算プロセッサ１２が独自にプログラムを持ち、各演算プロセッサ１２が独立して動作する方式も提案されており、ＭＩＭＤ（MultipleInstructionMultiple Data Stream）方式と呼ばれる。

【００１４】図３７に示したＳＩＭＤ方式は、ＭＩＭＤ方式に比較して制御が簡単でハードウェアが少なくてすみ、多くのデータに対して同じ処理を行う必要のある場合に有効である。しかし、データによって異なった処理が必要となる場合に、効率よく処理できないという欠点があり、この場合はＭＩＭＤ方式が有利になる。

【００１５】このような、並列計算機においては、各演算プロセッサ（演算要素／以下、ＰＥ：Ｐｒｏｃｅｓｓｉｎｇ－Ｅｌｅｍｅｎｔという）間の接続方式がその並列計算機をどのようにするかがその並列計算機の特徴を出す大きな要因となっている。ＰＥ間接続方式は大きく分けて、静的結合と動的結合の２つがある。

【００１６】静的結合は、隣接する通信相手のＰＥが経路によって固定されているもので、リング結合、トーラス結合、木構造結合などがある。その特徴は、通信経路が確定しているため、隣接経路のあるＰＥ間でのデータ通信は速いが、逆に遠くのＰＥ、すなわち他のＰＥを経由して通信を行わなければならない相手との通信は非常に遅くなるという問題がある。

【００１７】動的結合は、通信相手が固定でなく、クロスバスイッチを多段に設けて通信時にスイッチを切り替え、通信相手を決定するもので、クロスバネット、オメガネットなどがある。その特徴は、スイッチ群を経由することにより、ほぼ全てのＰＥへの通信距離が等価に見えることであり、任意のＰＥ間の通信時間がほぼ一定となることである。

【００１８】また、ＰＥが故障した場合の故障ＰＥの切り放しも容易に行うことが出来る。しかし、経路の確定のためと、スイッチのディレイのために、通信のための時間がかかり、また、スイッチでのデータの衝突などが問題となる。

【００１９】これらの結合路の使い分けとして、隣接作用問題でよく行われる隣のＰＥへのデータ転送では静的結合である隣接結合路を使用し、任意のＰＥから他の任意のＰＥへデータを転送するいわゆるランダム通信では動的結合の通信経路を使用する場合が多い。

【００２０】静的結合網の並列計算機に、サイズの合う問題を実行させる場合は効率はよいが、サイズの異なる問題を実行させる場合は、ＰＥ台数が中途半端になり、非常に効率が落ちてしまう。例えば３２×３２の２次元トーラス結合の並列計算機に問題サイズ３２×３２×１０の隣接作用問題を乗せる場合、ｘ軸、ｙ軸をそれぞれＰＥアレイの各辺に対応させれば効率は非常によい。

【００２１】ところが、問題サイズ３０×３０×１０の問題では、ｘ，ｙ軸をＰＥアレイの各辺に対応させてもそれぞれ２行分のＰＥ列が余ってしまう。この場合は、各２行分が余ったことをプログラマーあるいはコンパイラが認識してこれに見合ったコードを出力しなければならない。

【００２２】例えばリング状に隣接ＰＥに接続されている場合、問題サイズのために使用しないＰＥがあれば、同ＰＥを通過させるために、余分な転送動作を行う必要があり、各々のＰＥでは使用するＰＥと使用しないＰＥの判別動作も必要となってくる。そのため問題サイズとアレイサイズが一致した場合に比べ、データ転送でのオーバーヘッドにより明らかに処理効率が落ちる。

【００２３】並列計算機の他の特徴としては、各ＰＥの演算データを保持するメモリの構成も挙げられる。大きく分けると。共有メモリ型と、分散メモリ型がある。共有メモリ型は、各ＰＥとメモリ間を共有バスあるいは動的結合路で結合する場合が多く、比較的プロセッサ台数が少ない場合に有効である。

【００２４】一方、分散メモリ型は各々のＰＥにローカルメモリを付け、データの共有性を無くするかわりにデータアクセスのコンフリクトを解消している。そのため、近年のＰＥ台数の多い並列計算機では、ほぼこの分散メモリ型で構成されている。

【００２５】分散方式のローカルメモリでは、解く問題の並列度は低いが１台のＰＥで分担するデータ量が多い場合、一度に乗せることのできるデータ量は１台のＰＥのローカルメモリに依存してしまう。稼動しているＰＥよりもＩＤＬＥのＰＥの方が多い場合、本来１台のＰＥで受け持つデータを２台のＰＥに振り分け、２台のＰＥの連動により、疑似的に１台のＰＥとして処理を行う。この場合、２台の連動を処理するためのソフトウェアのオーバーヘッドと、隣接通信が行えないオーバーヘッドから、処理能力が低下する。

【００２６】以上のように、従来の並列計算機では問題サイズが異なることによるオーバーヘッドが大きいため、以前よりオーバーヘッドを小さくするような並列計算機が必要とされていた。

【００２７】一方、複数の演算プロセッサを相互結合する通信ネットワークは、これらの演算プロセッサを相互に結合するが、プロセッサの数は非常に多いので、任意の演算プロセッサから他の全ての演算プロセッサに結合するための通信経路を備えるのは、物理的にも経済的にも現実的ではない。

【００２８】そこで、例えば図１９に示すように、相互に隣接する演算プロセッサ５５間の通信チャネル５６だけを備えさせ、隣接せず直接に結合されていない演算プロセッサ５５間では、一つ以上の中継プロセッサを介して通信を行う方法がある。

【００２９】このようなプロセッサ間通信方式としては、例えばW.J.Dally 他の" Deadlock-Free Message Routing in Multiprocessor Interconnection Networks" (IEEETrans. Comput., vol.C-6, No.5, May 1987)に開示されているワームホール・ルーティングがある。このワームホール・ルーティングでは、メッセージをフリットと呼ばれる小さな単位に分割して通信を行う。通信は送信側プロセッサから受信側プロセッサまでの経路を確定して開始され、その経路は通信が終了するまでそのメッセージに占有される。

【００３０】ワームホール・ルーティングは以上のような通信を行うので、あるメッセージの通信が行われている間は、その通信で使用されている通信チャネルを他のメッセージの通信に使用することはできない。したがって、あるメッセージを構成する全てのフリットの通信が途中で停止してしまうと、その通信に使用されている通信チャネルを必要とする通信は、その間ずっと待ち続けなければならない。すなわち、あるメッセージの通信が他のメッセージの通信を妨害してしまうという問題がある。

【００３１】この問題を解決するために、例えばW.J.Dally の" Virtual-Channel Flow Control" (IEEE Trans. Para. and Dist. Sys., vol.3, No.2, Mar. 1992)に開示されているように、各演算プロセッサのメッセージの受信側に複数個のフリットを蓄えることのできるバッファを設ける。

【００３２】こうすることで、通信が停止しているメッセージを演算プロセッサ内に蓄え、これがネットワークに広がる範囲を狭め、このメッセージがさらに他のメッセージの通信を妨害しないようにしてる。

【００３３】このような演算プロセッサの従来例を図３８に示す。ルータ部５１と演算処理部５２から構成される。ルータ部５１はフリットを蓄えるバッファ５３ｘ，５３ｙ，５３ｐと、フリットの出力先を決定するスイッチ５４から構成される。

【００３４】Ｐｉｎ，Ｐｏｕｔはルータ部５１と演算処理部５２とを接続する通信チャネルである。Ｘｉｎ，Ｘｏｕｔ，Ｙｉｎ，Ｙｏｕｔはそれぞれ隣接する演算プロセッサとの間の入力および出力のための通信チャネルである。

【００３５】しかしこのプロセッサを用いた場合、以下のような問題がある。図３９において、５５ａ～５５ｅは演算プロセッサであり、それぞれバッファ５３ｘ，５３ｙ，５３ｐ、スイッチ５４および演算処理部５２から構成される。

【００３６】出力先の演算プロセッサ５５ｂのバッファが一杯で、通信が停止しているフリット６７が演算プロセッサ５５ａのバッファ５３ｘに蓄えられているとする。このとき演算プロセッサ５５ｃから演算プロセッサ５５ａに、別のメッセージのフリット６８が入力されるとする。このメッセージの出力先が演算プロセッサ５５ｅであるとすると、演算プロセッサ５５ｅのバッファに空きがあれば、フリット６８は出力されるべきである。

【００３７】ところが、バッファ５３ｘに蓄えられているフリット６７の通信が停止しているため、フリット６８を出力することができない。すなわち、出力先の通信チャネルが空いており本来出力されるべきであるメッセージが、通信の停止している他のメッセージによって通信を妨害される。したがって、並列計算機全体としての通信時間は増大し、スループットも低下する。

【００３８】ところで、図１９で示した通信方法とは別に、図２９に示すような、通信方法がある。この方法におけるワームホール・ルーティングでは、メッセージをフリットと呼ばれる小さな単位に分割し、中継プロセッサはメッセージが全て到達するのを待たないで、あるフリットが到着したら直ちにそのフリットを次のプロセッサに転送する。

【００３９】ワームホール・ルーティングを行うプロセッサの構成例を図４０に示す。９０はルータ部で、Ｘｉｎ、Ｘｏｕｔはそれぞれ隣接するプロセッサとの間の入

力および出力チャネルである。Ｐｉｎ、Ｐｏｕｔはルータ部９０と演算処理部７２を接続する通信チャネルである。

【００４０】図２９においてプロセッサ７４ａからプロセッサ７４ｅへの通信を考える。ワームホール・ルーティングではハンドシェイクを行って通信を行う必要がある。ハンドシェイクのリクエストとアクノリッジとで一回のフリットの通信が成立するので、一単位時間は２と考えることができる。Ｄをプロセッサ間の距離、Ｌをフリットの数として、図４１のＤ＝４、Ｌ＝４の場合を例にして、必要な通信時間を考察する。

【００４１】例えばプロセッサ７４ｂからプロセッサ７４ｃへ先頭フリットを送信している間に、プロセッサ７４ａはプロセッサ７４ｂに二番目のフリットを送信することができる。同様の考察から、プロセッサ７４ａからプロセッサ７４ｅまでの通信を考える。

【００４２】先頭フリットがプロセッサ７４ａからプロセッサ７４ｅに到達するまでに必要な時間は２×Ｄであり、この間に二番目のフリットは、プロセッサ７４ｄに到達している。したがって、二番目以降の全てのフリットが、プロセッサ７４ａから７４ｅに到達するまでに必要な時間は２×（Ｌ－１）となる。この結果、ワームホール・ルーティングの場合に必要なプロセッサ間の通信時間は２×（Ｄ＋Ｌ－１）となる。

【００４３】プロセッサ間通信の他の方式には、特開平２－２２８７６２に開示されているワームホール・ルーティングを変形した方法がある。この方法ではメッセージとともに伝達されるクロックの制御のもとに、プロセッサ間で通信が行われる。ワームホールでは通信を行うごとに、ハンドシェイクを行って隣接プロセッサ間で同期をとらなければならないのに対して、この方法ではいったんプロセッサ間の経路が決定してしまえば、クロック同期にしたがって通信すればよく、ハンドシェイクを行う必要がない。

【００４４】この方法ではいったん経路が確定した後では、ハンドシェイクを行う必要がないので（Ｄ＋Ｌ－１）で通信が行える。しかし、経路を確定するときと、経路の確定を送信側プロセッサへ報告するときには、ワームホール・ルーティングと同様にハンドシェイクを行わなければならず、それぞれ（２×Ｄ）の通信時間を必要とする。したがってワームホール・ルーティングの変形を用いた場合に必要なプロセッサ間の通信時間は（５×Ｄ＋Ｌ－１）となる。図４２にＤ＝４、Ｌ＝４の場合のタイミング図を示す。

【００４５】プロセッサの数が１，０００を越えるような大規模なネットワークになると、これらのＤの項による影響が無視できなくなる。このとこはＤが大きくなるとそのメッセージの通信に要する時間が増大するだけではない。すなわち、通信中のメッセージがプロセッサやプロセッサ間のチャネルを占有し、他のメッセージはそのプロセッサやチャネルを利用することができなくなる。

【００４６】そのためプロセッサやチャネルが解放されるのを待たなければならない。つまり、あるメッセージの通信が他のメッセージの通信の妨げとなるわけである。したがってメッセージの通信に時間がかかると、ほかのメッセージの障害となる時間も増加し、ネットワーク全体での通信時間はさらに増大することになる。

【００４７】

【発明が解決しようとする課題】以上で述べたように、従来のＳＩＭＤ型並列計算機では、問題の並列度がプロセッサよりも大きい場合には、その並列度を生かして高速に問題を解くことができるが、問題の並列度がプロセッサ数よりも小さい場合には、有効に利用できるのは問題の並列度分のプロセッサのみで、それ以外のプロセッサは遊ぶことになり、問題を解く時間を短縮するのには役にたたないという問題がある。

【００４８】また、ＳＩＭＤ型並列計算機は、多くのデータに対して同じ処理を行う必要のある場合に有効である。しかし、データによって、異なった処理が必要となる場合に、効率よく処理できないという欠点があった。一方ＭＩＭＤ型並列計算機では、データによって異なる処理が必要な場合も柔軟に対応できるが、制御が複雑で、ハードウェアが増大するという欠点があった。

【００４９】さらに、従来の静的結合網の並列計算機では、問題のサイズが異なる場合にＰＥ台数がが中途半端になり、効率が低下してしまったり、データ量が多い場合に複数台数のＰＥを連動させるためのオーバーヘッドが大きくなるという問題があった。

【００５０】一方、従来の並列計算機では、出力先の通信チャネルが空いており、通信が可能であるにも関わらず、通信が停止しているメッセージによって他のメッセージの通信が妨害されてしまうという問題があった。

【００５１】また、ネットワークの規模がプロセッサ間通信に与える影響が大きく、大規模なネットワークでは通信に非常に時間がかかってしまうという問題もあった。

【００５２】この問題を解決するために、第１の発明は、問題の並列度がプロセッサ数よりも小さい場合でも、できるだけ多くのプロセッサを効率よく利用できる並列計算機を提供することを目的とする。

【００５３】第２の発明は、ＳＩＭＤ型並列計算機に若干のハードウェアを付加するだけで、ＭＩＭＤ処理を可能とする並列計算機を提供することを目的とする。

【００５４】第３の発明は、問題のサイズが異なる場合に、使用されないＰＥをバイパスすることによって処理効率の低下を防ぎ、さらにデータ量が多い場合に、使用されないＰＥのローカルメモリを使用されるＰＥからアクセス可能にすることによってオーバーヘッドを小さくすることができる並列計算機を提供することを目的とす

る。

【００５５】また、第４の発明は、通信可能なメッセージが妨害されずに、通信時間の短縮を可能とする並列計算機を提供することを目的とする。

【００５６】第５の発明は、通信チャネルおよび制御信号を中継プロセッサでバイパスすることにより、通信時間の短縮を可能とする並列計算機を提供することを目的とする。

【００５７】

【課題を解決するための手段】上記目的を達成するため、第１の発明は、複数個のプロセッサからなるプロセッサアレイと、そのプロセッサアレイに対して並列実行可能な複数種類の命令列を抽出して供給するなどの制御を行なうプロセッサアレイ制御手段から構成される。

【００５８】各プロセッサは、プロセッサアレイ制御手段から供給される複数種類の命令列の中の１種類ずつを受け取る機能と、受け取った命令列を他のプロセッサと並列に実行する機能とを持つ。

【００５９】また、第２の発明の並列計算機は、大きく分けて、複数個の演算プロセッサと、これを制御する単一の制御プロセッサとから構成される。さらに、演算プロセッサは、演算プロセッサでの実行を制御する命令を格納するためのレジスタと、このレジスタに、制御プロセッサからの命令を入力するか、あらかじめメモリに格納されている命令を格納するかを選択するためのマルチプレクサと、レジスタに格納されている命令にしたがって演算を実行する演算部とから構成される。

【００６０】第３の発明は、個々の通信経路で接続されている複数の演算要素と、各々の演算要素と個々に結合しているローカルメモリと、解く問題のサイズに応じて演算に使用する演算要素と使用しない演算要素とを識別する識別手段と、この識別手段によって識別された、演算に使用しない演算要素をバイパスするバイパス手段と、前記識別手段によって識別された、演算に使用しない演算要素と結合しているローカルメモリを、演算に使用する演算要素から直接アクセス可能にするアクセス手段とから構成されている。

【００６１】さらに、第３の発明は、前記複数の演算要素、識別手段、バイパス手段、ローカルメモリ、及びアクセス手段を１つのチップ上に配置し、前記バイパス手段及びアクセス手段を前記演算要素上に配線する。

【００６２】また、第４の発明は、複数のプロセッサがＮ次元ネットワークで相互接続され、各プロセッサが、隣接するプロセッサからメッセージを受信し、出力先を決定する出力先決定手段と、出力先ごとに設けられ、前記出力先決定手段によって出力先が決定されたメッセージを受信した順に蓄える蓄積手段と、蓄積されたメッセージを、決定された出力先のプロセッサに出力する出力手段とから構成される。

【００６３】第５の発明は、複数のプロセッサがＮ次元ネットワークで相互接続され、各プロセッサが、隣接するプロセッサからメッセージおよび制御信号を受信し、メッセージの宛先を判定する判定手段と、判定されたメッセージが自プロセッサ以外の場合、受信したメッセージおよび制御信号を隣接するプロセッサにバイパスするバイパス手段とから構成されている。

【００６４】

【作用】上記手段により、第１の発明は、対象とする問題の並列度が十分大きい場合には、従来のＳＩＭＤ型並列計算機と同様の動作をさせる。その場合には、プロセッサアレイ制御手段が、プロセッサアレイへ供給する複数個の命令列をすべて同一の命令列にすることによって実現できる。各プロセッサは、プロセッサアレイ制御手段が供給する複数個の命令列の一つを受け取り実行することができるから、複数個の命令列がすべて常に同一であれば、すべてのプロセッサが常に同一命令を並列実行することになる。

【００６５】対象とする問題の並列度がプロセッサ数より小さい場合には、以下のように制御する。ここで、プロセッサアレイ制御手段は、並列実行可能なm種類の命令列を抽出してプロセッサアレイへ供給できるようになっており、プロセッサはｎ個あるものとする。そして、ｎ個のプロセッサをm個ずつのｋ組に分ける（すなわち、ｎ＝ｍ×ｋ、ｎ、ｍ、ｋは整数とする）。このm個のプロセッサからなる組をクラスタと呼ぶことにする。

【００６６】各クラスタにはすべて同じm種類の命令列を供給し、同じ処理を行なうように制御する。プロセッサアレイ制御手段から供給されるm種類の命令列は、各クラスタ内のm個のプロセッサのそれぞれに１種類ずつ供給され、各プロセッサは、供給された命令列を実行する。

【００６７】また、第２の発明は、通常のＳＩＭＤ処理では、各演算プロセッサは、制御プロセッサから放送される命令にしたがって処理を行う。制御プロセッサから指示があると、あらかじめ各演算プロセッサのローカルメモリに格納されている命令が命令レジスタに格納されるようにマルチプレクサが切り換わる。各演算プロセッサのローカルメモリに異なる命令を格納しておくことによって、ＭＩＭＤ処理を行う。

【００６８】さらに、第３の発明は、ＰＥアレイのサイズと問題サイズが異なる場合、コンパイラあるいは他の手段により、使用するＰＥと、使用しないＰＥを予め決定し、決定されたＰＥ情報を識別手段に送る。識別手段では、ＰＥ毎の使用する／しないの情報を保持しておく。

【００６９】処理実行時に隣接通信を行う際、各ＰＥのバイパス手段は自ＰＥに対応する識別手段から、バイパスを行うかどうかが指示される。稼動中のあるＰＥは、隣のＰＥが休止である場合、休止のＰＥのバイパス手段により、さらに先の稼動中のＰＥがあたかも隣のＰＥに

見える。
【００７０】以上から、処理する問題のサイズに合わせて使用するＰＥと使用しないＰＥを適切に選択することにより、解く問題に最適なハードウェアのアレイサイズを用意する。
【００７１】また、アクセス手段に予め休止となるＰＥのローカルメモリをアクセスする稼動ＰＥの情報を与えておき、同情報と識別手段の情報から、稼動するＰＥは休止しているＰＥのローカルメモリに直接アクセスし、同ローカルメモリを稼動するＰＥのローカルメモリの一部として使用する。
【００７２】さらに、バイパス手段とアクセス手段を、同一チップ内に納められた複数ＰＥに適用する事により、高速な動作を保証すると共に、バイパス手段及びアクセス手段をＰＥの上に配線することにより配線面積を小さくし、チップ面積の縮小を行う。
【００７３】第４の発明の並列計算機によれば、メッセージの出力先の決定後にフリットをバッファに蓄えるので、出力先チャネルが空いていれば、通信が停止している他のメッセージによってそのメッセージの通信が妨害されることはない。したがって、並列計算機全体として通信時間は短縮され、スループットは向上する。
【００７４】第５の発明の並列計算機によれば、あるプロセッサが隣接するプロセッサからメッセージを受け取り、そのメッセージが自プロセッサ宛でなければ、通信チャネルと制御信号をバイパスする。メッセージの中継を行うプロセッサで、メッセージおよび制御信号をバイパスし、通信を行っているプロセッサ間の距離を実際よりも小さくしている。
【００７５】
【実施例】以下、図面を参照しながら本発明の実施例を説明する。
【００７６】第１の発明
第１の発明の一実施例の構成を図１に示す。図１に示した並列計算機は、ｎ個（ｎは整数）のプロセッサ１からなるプロセッサアレイと、プロセッサアレイに対して命令を供給するプロセッサアレイ制御装置２とからなる。図示していないが、プロセッサ１間およびプロセッサ１とプロセッサアレイ制御装置２間には、データ転送と制御のための通信ネットワークが存在する。
【００７７】なお、今回の実施例では、プロセッサアレイ制御装置２が２種類の命令列を供給する場合について示すが、第１の発明はこれに限らず、複数種類の命令列を供給することができるものである。
【００７８】プロセッサアレイ制御装置２は、命令メモリを持ち、そこから命令を取り出し、プロセッサアレイに対して１度に２個の命令を供給することができる。各プロセッサ１は、プロセッサアレイ制御装置２から同時に供給された２個の命令のうちの１つの命令を受け取り実行する。
【００７９】具体的には、プロセッサ番号が偶数のプロセッサ１はプロセッサアレイ制御装置２の出力端子ＯＵＴ０から出力される命令列の命令を受け取り実行し、プロセッサ番号が奇数のプロセッサ１は出力端子ＯＵＴ１から出力される命令列の命令を受け取り実行する。
【００８０】各プロセッサ１は、演算部、レジスタファイル、制御部、ローカルメモリ、メモリインターフェース部、プロセッサ間通信部からなる。これらのプロセッサ１には２つの動作モードがある。１つは、各プロセッサ１が独立に動作するレジスタ非共有モードであり、もう１つは、隣のプロセッサ１とレジスタを共有するレジスタ共有モードである。
【００８１】ここでいうレジスタの共有とは、自分のプロセッサ１内のレジスタの内容が対応する相手のレジスタの内容と常に同じようになっているということである。つまり相手のプロセッサ１内のレジスタにデータを書き込んだ後に、自分の対応するレジスタを読み出すと相手のプロセッサ１が書き込んだデータと同じデータが読み出せるというものである。
【００８２】プロセッサ番号が偶数のプロセッサ１は、自分のプロセッサ番号より１つ大きい番号をもつプロセッサ１とレジスタを共有し、プロセッサ番号が奇数のプロセッサ１は、自分のプロセッサ番号より１つ小さいプロセッサ１とレジスタを共有する。
【００８３】まず、プロセッサ数よりも並列度が大きい処理を実行する場合について説明する。この場合には、プロセッサ１の動作モードはレジスタ非共有モードとし、すべてのプロセッサ１に同じ処理を実行させるために同じ命令を供給するようにする。そのためには、この実施例では、プロセッサ１ごとに供給される命令列は固定されているので、プロセッサアレイ制御装置２の２つの出力端子ＯＵＴ０，１から同一の命令列を供給することにより、１つの命令列を２つの命令列として供給するように制御する。
【００８４】次にプロセッサ数よりも並列度が小さい処理を実行する場合について説明する。この場合には、プロセッサ１の動作モードはレジスタ共有モードとする。プロセッサ番号が偶数のプロセッサ１とその番号より１つ大きい番号のプロセッサ１とが１つのクラスタを構成する。
【００８５】クラスタ内の２つのプロセッサ１には、プロセッサアレイ制御装置２によって抽出された並列実行可能な２つの命令が出力端子ＯＵＴ０，１からそれぞれのプロセッサ１に同時に供給され、それぞれのプロセッサ１では供給された各命令を並列に実行する。
【００８６】ここで、図２のような式の計算をする場合について説明する。例えば、プロセッサ数ｎが１０００であるとする。
【００８７】この計算を１０００組のデータに対して行なう場合には、プロセッサ１のモードをレジスタ非共有

モードにし、プロセッサアレイ制御装置２の出力端子ＯＵＴ０，１から、図３（ａ），（ｂ）のような同一な命令列１，２を供給する。各プロセッサ１に１組ずつのデータの処理を割当てることにより、１個のプロセッサ１で１０００組のデータを処理するのに比べ１０００分の１の時間で処理できる。

【００８８】しかし、このレジスタ共有モードで５００組のデータを処理する場合は、１個のプロセッサ１で５００組のデータを処理するのと比べ、プロセッサ数が１０００であるのに５００分の１の時間でしかなく、並列計算機の能力を十分に生かしきれない。

【００８９】一方、プロセッサ１のモードをレジスタ共有モードにし、プロセッサアレイ制御装置２から図４（ａ），（ｂ）のような並列実行可能な２種類の命令列１，２を出力端子ＯＵＴ０，１からそれぞれ供給すると、２台のプロセッサ１で１組のデータを処理することになり、１０００分の１の時間で実行することが可能である。

【００９０】図３の命令列では、１回の処理を行なうのに４命令分実行時間がかかるが、図４の場合には、２命令分の実行時間ですみ、実行時間が半分になる。したがって、５００組のデータに対する処理の場合でも図４の命令列で実行させれば、１個のプロセッサ１で処理する場合に比べて１０００分の１の時間で実行することが可能である。

【００９１】なお、プロセッサアレイ制御装置２が供給する並列実行可能な複数の命令は、もともとメモリ上に並列実行可能な複数の命令列を用意しておき、それを供給するという方法と、単一の命令列を用意しプロセッサアレイ制御装置２内で並列実行可能な命令を抽出して複数命令を供給するという方法が考えられるがどちらでも良い。後者の場合には、通常のＳＩＭＤ型並列計算機とのオブジェクト互換性が保たれることになるため、その点は有利である。

【００９２】図１に示した実施例では、複数供給される命令列のうちのどちらを各プロセッサ１が受け取るかが固定されているが、図５に示すように、いずれの命令列を受け取るかをセレクタ３で選択できるようにする構成もある。この構成では、すべてのプロセッサ１は、プロセッサアレイ制御装置２から供給される２つの命令列のどちらでも選択して受け取ることができる。

【００９３】その選択は各プロセッサ１の命令入力の前におかれているセレクタ３で行なわれ、どちらを選ぶかは、あらかじめ設定することもできるし、各プロセッサ１の演算結果からどちらを選択するかを決定することもできる。クラスタを構成する際には、並列実行可能な２つの命令列を受け取った２つのプロセッサ１で構成すれば良い。

【００９４】この場合、ｉｆ－ｔｈｅｎ－ｅｌｓｅのような処理を実行するときに、ｔｈｅｎ処理とｅｌｓｅ処理を並列に実行するように２つの命令列を出力端子ＯＵＴ０，１からそれぞれ供給し、各プロセッサ１はｉｆ文の条件によりいずれかの命令列を選択するようにすれば、ｉｆ－ｔｈｅｎ－ｅｌｓｅの並列実行が可能である。

【００９５】ｓｕｐｅｒｓｃａｌａｒあるいはＶＬＩＷなどの複数命令の並列実行機構を持ったプロセッサなどを要素プロセッサとして用いる場合には、並列度の高い問題を解く場合でも、問題の並列度を犠牲にして複数命令の並列実行をすることになり、必ずしも効果があがるとはかぎらないが、この第１の発明では、並列度の高い問題では、問題の並列性を利用して解くことができるので、高速に解くことが可能である。

【００９６】第２の発明

図６は、第２の発明の並列計算機に係わる一実施例の構成を示すブロック図である。

【００９７】同図に示す並列計算機は、大きく分けて、１台の制御プロセッサ（ＣＰ）１１と、複数台の演算プロセッサ（ＰＥ）１２から構成される。制御プロセッサ１１には全体メモリ（Ｍ）１３が接続され、演算プロセッサ１２にはそれぞれローカルメモリ（ＬＭ）１４が接続される。制御プロセッサ１１と各演算プロセッサ１２は、グローバルバス１５、および同期バス１７で接続される。演算プロセッサ１２は、ネットワーク１６で相互接続される。ネットワーク１６の形態としては、格子結合、２進ｎキューブ結合、木結合等、多くのものが知られており、いずれの方式でもよい。

【００９８】制御プロセッサ１１は、それ自体が一つの情報処理装置であり、全体メモリ１３に格納されているプログラム（命令列）を読み込み、解読し、解読結果にしたがって、全体メモリ１３に格納されているデータに対して、処理を行う機能を有する。さらに制御プロセッサ１１には、演算プロセッサ１２に対して制御を行う機能が追加されている。

【００９９】全体メモリ１３に格納されている制御プロセッサ１１のプログラムには、制御プロセッサ１１自身での処理を指示する命令の他に、演算プロセッサ１２での処理を指示する命令が含まれている。

【０１００】制御プロセッサ１１は、メモリ１３から読み出した命令を解読し、制御プロセッサ１１で実行すべき命令であれば、制御プロセッサ１１で実行する。解読した命令が演算プロセッサ１２で実行すべき命令であれば、グローバルバス１５を介して全演算プロセッサ１２に放送する。ここで放送する命令は、解読済みのもの（マイクロ命令）でも解読前のもの（機械語命令）でもよいが、ここでは解読済みの命令を放送するものとして話を進める。

【０１０１】制御プロセッサ１１は、グローバルバス１５を介して、すべての、あるいは特定の演算プロセッサ１２から、データの読み出し、および書き込みを行うこ

とができる。また、同期バス１７を介して、すべての演算プロセッサ１２の状態の論理和を認識することができる。

【０１０２】さらに制御プロセッサ１１は、フロントエンド計算機１８と結合されている。フロントエンド計算機１８では、制御プロセッサ１１および演算プロセッサ１２で実行するプログラムの開発を行ったり、処理すべきデータの入出力を行う。

【０１０３】フロントエンド計算機１８と制御プロセッサ１１は、実行時、制御プロセッサ１１で実行するべきプログラムをフロントエンド計算機１８から全体メモリ１３に転送する。さらに、制御プロセッサ１１で実行すべきデータをフロントエンド計算機１８から全体メモリ１３に転送したり、演算プロセッサ１２で実行するべきデータを、フロントエンド計算機１８から各演算プロセッサ１２に分配したり、あるいは処理結果を各演算プロセッサ１２や全体メモリ１３からフロントエンド計算機１８に転送したりする機能を有する。

【０１０４】演算プロセッサ１２は、制御プロセッサ１１からグローバルバス１５を介して放送される命令にしたがって、自ローカルメモリ１４のデータに対して演算を行ったり、ネットワーク１６を介して他の演算プロセッサ１２とデータ通信を行ったりして処理を行う。

【０１０５】第２の発明の中心となる演算プロセッサ１２は、図７に示すような構成になっている。

【０１０６】まず、ローカルメモリ１４と演算プロセッサ１２とは、メモリアドレスバス２１と、メモリデータバス２２によって接続されている。

【０１０７】マルチプレクサ（ＭＵＸ）２３は、第２の発明において追加されたものであり、命令レジスタ２４に格納する命令として、制御プロセッサ１１からグローバルバス１５を介して放送される命令か、ローカルメモリ１４から読み出されたデータかを選択する。マルチプレクサ２３の一方の入力は、制御プロセッサ１１からのグローバルバス１５が、もう一方の入力には、メモリデータバス２２がそれぞれ接続されている。

【０１０８】命令レジスタ２４には、マルチプレクサ２３で選択された命令が格納される。演算プロセッサ１２を構成する他のすべての要素は、命令レジスタ２４からの指示によって制御される。ここでは説明を容易にするために、命令レジスタ２４の最上位ビットでマルチプレクサ２３の制御を行うこととし、最上位ビットが０ならば、制御プロセッサ１１から放送される命令が命令レジスタ２４に格納され、最上位ビットが１ならば、ローカルメモリ１４から読み出されるデータが命令レジスタ２４に格納されるものと仮定して話を進める。

【０１０９】プログラムカウンタ（ＰＣ）２５は、第２の発明をより効果的にするために追加されたものであり、ローカルメモリ１４に格納されているＭＩＭＤ処理用の命令列を読み出すために用いられる。プログラムカウンタにＭＩＭＤ用命令列のアドレスを格納し、読み出し操作を行うことによって、メモリデータバス２２に命令が読み出される。命令列を連続して読み出すために、プログラムカウンタ２５には、自分自身の値をインクリメントする機能を持つ。

【０１１０】アドレスレジスタ（ＡＲ）２６、データレジスタ（ＤＲ）２７は、ローカルメモリ１４をアクセスするために用いられる。アドレスレジスタ２６は、内部バス２８から書き込まれ、メモリアドレスバス２１に値を出力する。データレジスタ２７は、メモリデータバス２２と内部バス２８との間に置かれる双方向のレジスタである。

【０１１１】データ読み出し時は、内部バス２８からアドレスレジスタ２６に、読み出すべきメモリ１３のアドレスが格納され、ローカルメモリ１４に対して読み出し操作を行い、読み出されたデータはメモリデータバス２２からデータレジスタ２７に格納される。データ書き込み時は、アドレスレジスタ２６に書き込むべきメモリ１３のアドレスを格納し、データレジスタ２７に書き込みデータを格納し、書き込み操作を行うことにより、書き込みが行われる。

【０１１２】ＡＬＵ２９は、内部バス２８を介して、レジスタファイル３０に格納されるデータに対して、算術あるいは論理演算を行う。

【０１１３】状態制御レジスタ（ＰＳＷ：Prosessor Status Word）３１は、ＡＬＵ２９での演算状態を表示するビットと、演算プロセッサ１２での演算を制御するビットから構成される。演算を制御するビットの内訳は次の通りである。このうち、（１）～（３）は、演算のＡＬＵ２９で演算が行われるたびに設定される。（４），（５）は、直接、状態制御レジスタ３１を書き換えることによって設定する。

【０１１４】（１）ゼロビット：演算結果がゼロになるとセットされる。

【０１１５】（２）正ビット：演算結果が正のときにセットされる。

【０１１６】（３）オーバーフロービット：演算でオーバーフローが発生したときにセットされる。

【０１１７】（４）同期ビット：全演算プロセッサ１２の同期ビットはワイアード・オア接続され、制御プロセッサ１１に接続される。同期を取る時点で演算プロセッサ１２がこのビットをセットすると、制御プロセッサ１１は、全演算プロセッサ１２が同期ビットをセットしたか否かを認識できる。

【０１１８】（５）マスクビット：このビットがセットされると、以後、制御プロセッサ１１からマスク解除命令が発行されるまで、命令レジスタ２４の内容にかかわらず、演算プロセッサ１２は動作しない。

【０１１９】プロセッサ番号レジスタ（ＰＮＲ：Porcessor Number Register）３２は、各演算プロセッサ固有

のプロセッサ番号が格納されるレジスタである。このレジスタを読み出すことによつて、自分のプロセッサ番号を知ることができる。このレジスタはハードウェア的に設定する方法や、システム立ち上がり時に制御プロセッサ１１が設定する方法等が考えられる。ここではハードウェア的に設定されているものとするが、他の方法でも差し支えない。

【０１２０】通信制御部３３は、ネットワーク１６を介して他の演算プロセッサ１２と通信したり、グローバルバス１５を介して制御プロセッサ１１との通信を行うためのレジスタや外部の通信プロトコル制御装置から構成される。演算プロセッサ１２は、この通信制御部３３によって、制御プロセッサ１１や他の演算プロセッサ１２と通信を行うことができる。

【０１２１】命令レジスタ２４への命令の格納は、前述したように、次の２通りの方法で行われる。第一の方法は、制御プロセッサ１１より放送されるものである。制御プロセッサ１１から全演算プロセッサ１２に同一の命令が放送されるため、マスクビットがセットされていない演算プロセッサ１２は、すべて同じ処理を行う。すなわち、ＳＩＭＤ方式の並列処理が行われる。

【０１２２】第２の方法は、ローカルメモリ１４にあらかじめ格納されている命令を読み出して、命令レジスタ２４に格納するものである。この命令は、あらかじめ各演算プロセッサ１２で異なったものを格納しておくことにより、各演算プロセッサ１２で異なった処理を行うことができる。すなわち、ＭＩＭＤ方式の並列処理となる。

【０１２３】以下では、ＳＩＭＤ方式とＭＩＭＤ方式の２つの処理方式、およびこの２つの処理方式間の以降を中心に、本並列計算機での処理手順について説明する。

【０１２４】まず、第２の発明の並列計算機で実行するプログラムについて説明する。このプログラムは、大きく次の３つに分けられる

（１）逐次処理される部分。

【０１２５】ループ制御等、並列処理できない部分で、制御プロセッサ１１で実行される。

（２）各演算プロセッサ１２が一斉に処理を行う部分（ＳＩＭＤ処理）。

【０１２６】制御プロセッサ１１により、各演算プロセッサ１２が同じ処理を行う。

【０１２７】（３）各演算プロセッサ１２で異なった処理を行う部分（ＭＩＭＤ処理）。

【０１２８】制御プロセッサ１１の制御を離れて、各演算プロセッサ１２が独自に実行する。

【０１２９】このうち、（１）と（２）は、制御プロセッサ用プログラムとしてコンパイルされ、機械語命令列が全体メモリ１３にロードされる。そして、制御プロセッサ１１が解読、実行あるいは実行制御を行う。（３）は各演算プロセッサ用としてコンパイルされ、さらにマイクロ命令列に展開された後、実行される演算プロセッサ１２のローカルメモリ１４にロードされる。

【０１３０】次に、初期データのマッピング処理について説明する。

【０１３１】全体メモリ１３に、制御プロセッサ１１で実行される命令とデータがフロントエンド計算機１８から転送される。

【０１３２】次に、各演算プロセッサ１２のローカルメモリ１４に、演算データと、ＭＩＭＤ動作されるべき命令とが格納される。これらのデータは、各演算プロセッサ１２によって異なるため、演算プロセッサ１２ごとに逐次的に行われる。処理手順は次の通りである。

【０１３３】（１）制御プロセッサ１１は、全演算プロセッサ１２に対して、データを転送する演算プロセッサ番号と、マスク命令を放送する。

【０１３４】（２）各演算プロセッサ１２は、制御プロセッサ１１からの命令に従って、制御プロセッサ１１から放送されるプロセッサ番号と、自分のプロセッサ番号とを比較し、一致していなければマスクビットをセットする。したがって以後の処理は、制御プロセッサ１１が指定した番号の演算プロセッサ１２のみで行われる。

【０１３５】（３）制御プロセッサ１１は、データを格納するべきローカルメモリ１４の領域の先頭アドレスを演算プロセッサ１２に転送し、ＤＭＡモードと、制御プロセッサ１１から転送するデータのメモリ１３への書き込みを指示する命令を発行する。

【０１３６】（４）制御プロセッサ１１は、指定した演算プロセッサ１２に転送するデータをフロントエンド計算機１８から取得し、グローバルバス１５を用いて演算プロセッサ１２に順次転送する。

【０１３７】（５）指定された演算プロセッサ１２は、制御プロセッサ１１から転送されるデータを、指定されたローカルメモリ１４の領域に、ＤＭＡモードで順次書き込む。

【０１３８】（６）転送が終了すると、制御プロセッサ１１は、ＤＭＡ中止命令と、マスク解除命令を全演算プロセッサ１２に対して発行する。

【０１３９】（７）以上で述べた（１）～（６）の処理を全演算プロセッサ１２に対して行う。

【０１４０】ＭＩＭＤ方式の並列処理を行うにあたって、ＭＩＭＤ用命令アドレステーブルを作成する必要がある。このアドレステーブルの作成方法を下記に示す。

【０１４１】ＭＩＭＤ動作される命令列は、初期データマッピング時は、演算データと区別されることなくローカルメモリ１４に格納される。連続した命令列は、ローカルメモリ１４の連続したアドレスに格納されることが望ましい。また、同一時刻にＭＩＭＤ処理される命令列の開始アドレスは、各演算プロセッサ１２で同じであることが望ましいが、異なっていても実行可能である。

【０１４２】ＭＩＭＤ動作される命令列の開始アドレス

は、各演算プロセッサ１２で同一の場合は制御プロセッサ１１または演算プロセッサ１２が、各演算プロセッサ１２で異なる場合は演算プロセッサ１２が認識しておく必要がある。

【０１４３】図８に、各演算プロセッサ１２でＭＩＭＤ動作命令開始アドレスが同一の場合の制御プロセッサ１１のアドレステーブルの一例を示す。アドレステーブルはどの領域に作成してもよいが、ここでは全体メモリ１３の０番地から作成されるものと仮定する。同一時期にＭＩＭＤ動作される命令列の開始アドレスが、各演算プロセッサ１２で異なる場合は、各演算プロセッサ１２がローカルメモリ１４中にアドレステーブルを作成する必要がある。

【０１４４】以下に、ＳＩＭＤ方式による処理を説明する。

【０１４５】第２の発明の並列計算機における制御プロセッサ１１から放送される命令により、全演算プロセッサ１２が同一の処理を行う方式である。制御プロセッサ１１は、全体メモリ１３から命令を読み込み、解読した結果、それが演算プロセッサ１２で実行すべきものならば、グローバルバス１５を介して全演算プロセッサ１２に放送する。各演算プロセッサ１２では、制御プロセッサ１１から放送される命令を命令レジスタ２４に入力するように、マルチプレクサ２３を制御する。

【０１４６】マルチプレクサ２３の制御は、命令レジスタ２４の最上位ビットで行われ、最上位ビットが０ならば、制御プロセッサ１１から放送される命令が命令レジスタ２４に格納される。したがって、制御プロセッサ１１から放送されるＳＩＭＤ処理用の命令は、最上位ビットは常に‘０’となるように制御プロセッサ１１を設計する必要がある。命令レジスタ２４の各ビットが、演算プロセッサ１２の他の構成要素を制御することによって処理が行われる。

【０１４７】次に、ＳＩＭＤ方式からＭＩＭＤ方式へ移行する処理手順を説明する。

【０１４８】制御プロセッサ１１の命令には、各演算プロセッサ１２のローカルメモリ１４に格納されている命令列に実行を移行するための命令が用意されている。この命令は、図８に示すアドレステーブルの番号を指定するフィールドを持つ。この命令が制御プロセッサ１１で実行されると、次のような手順で各演算プロセッサ１２のＭＩＭＤ用命令列に処理が移行される。

【０１４９】（１）アドレステーブルから命令列の開始アドレス読み出され、グローバルバス１５を介して全演算プロセッサ１２に放送される。

【０１５０】（２）制御プロセッサ１１からの命令による制御で、各演算プロセッサ１２で、（１）で転送されたアドレスが、プログラムカウンタ２５に格納される。

【０１５１】（３）制御プロセッサ１１からの命令による制御で、各演算プロセッサ１２で、プログラムカウンタ２５で示されるローカルメモリ１４のアドレスから、ＭＩＭＤ処理のための命令が読み出される。このときの制御プロセッサ１１からの命令は、最上位ビットが‘１’になっている。したがって、次の命令レジスタ２４の入力は、ローカルメモリ１４から読み出されたＭＩＭＤ用命令が選択されるよう、マルチプレクサ２３が切り換わる。

【０１５２】（４）ローカルメモリ１４から読み出された命令は、命令レジスタ２４に格納され、命令レジスタ２４からの指示によって処理が行われる。この処理は、各演算プロセッサ１２で異なっていても良いため、ＭＩＭＤ処理となる。

【０１５３】次に、ＭＩＭＤ方式に移行した後の処理を以下に示す。

【０１５４】前述したように、ＭＩＭＤ処理が開始されたとき、プログラムカウンタ２５には、ＭＩＭＤ用命令列の先頭アドレスが格納されている。プログラムカウンタ２５は、命令レジスタ２４からの制御によってインクリメントされる機能を備えている。これによって、ローカルメモリ１４に格納されたＭＩＭＤ用命令列を順次読み出すことができる。

【０１５５】また、命令レジスタ２４に格納される命令の選択は、現在の命令レジスタ２４の最上位ビットで決定される。すなわち最上位ビットが‘１’ならば、ローカルメモリ１４から読み出した命令が命令レジスタ２４に格納される。

【０１５６】したがって、ローカルメモリ１４に格納されるＭＩＭＤ処理用命令列は、最上位ビットを‘１’にし、さらにプログラムカウンタ２５をインクリメントしながらローカルメモリ１４を読み出すように制御コードを設定しておけば、ローカルメモリ１４の命令列を順次命令レジスタ２４にロードして、ＭＩＭＤ処理が可能である。命令列の最後尾は、最上位ビットを‘０’にしておけば、ＭＩＭＤ処理終了後、制御プロセッサ１１からの命令が命令レジスタ２４にロードされるように、マルチプレクサ２３が制御される。

【０１５７】もちろん、単一の命令に、上記の最後尾の命令の設定を行っておけば、単一の命令のＭＩＭＤ動作も可能である。

【０１５８】最後に、ＭＩＭＤ方式からＳＩＭＤ方式への移行処理を説明する。

【０１５９】ＭＩＭＤ処理の終了は、各演算プロセッサ１２でまちまちであるため、制御プロセッサ１１は、全演算プロセッサ１２でのＭＩＭＤ処理の終了を確認してからＳＩＭＤ処理を開始する必要がある。

【０１６０】（１）制御プロセッサ１１は、各演算プロセッサ１２でのＭＩＭＤ処理を起動した後、待ち状態に入る。この状態では、制御プロセッサ１１は、演算プロセッサ１２に対してＮＯＰ（No Operation）コードを発行し続ける。

【０１６１】（２）演算プロセッサ１２はＭＩＭＤ処理を行うが、ＭＩＭＤ処理用命令列の最後尾の命令として、状態制御レジスタ３１の同期ビットをセットし、さらに最上位ビットを‘０’にした命令を挿入しておく。

【０１６２】（３）各演算プロセッサ１２では、ＭＩＭＤ処理用命令列の最後のコードを実行すると、状態制御レジスタ３１の同期ビットがセットされ、さらに命令レジスタ２４への入力が、以後は制御プロセッサ１１が発行する命令を選択するようにマルチプレクサ２３が切り換わる。各演算プロセッサ１２の同期ビットはワイヤード・オア接続され、制御プロセッサ１１は、全演算プロセッサ１２で同期ビットが１にセットされたか否かを認識できる。

【０１６３】（４）制御プロセッサ１１は、（１）で演算プロセッサ１２のＭＩＭＤ処理を起動した後、全演算プロセッサ１２で同期ビットに１がセットされたことを認識するまでＮＯＰ命令を発行し続ける。早く処理が終わった演算プロセッサ１２は、制御プロセッサ１１からＮＯＰコードを受け取って実行するが、副作用はない。

（５）制御プロセッサ１１は、全演算プロセッサ１２の同期ビットに１がセットされたこと、すなわち全演算プロセッサ１２でＭＩＭＤ処理が終了したことを確認したら、ＮＯＰコードの発行を中止し、ＳＩＭＤ処理用の命令の発行を開始する。

【０１６４】（６）以降は、制御プロセッサ１１からの命令にしたがって、全演算プロセッサ１２でＳＩＭＤ処理が行われる。

【０１６５】以上で述べたように、ＳＩＭＤ型並列計算機にわずかなハードウェア（マルチプレクサ２３，プログラムカウンタ２５）を追加するだけで、ＭＩＭＤ動作が可能となる。

【０１６６】なお、以上の実施例は、第２の発明を実現するための一例を示したものであり、これに限定されるものではない。例えば、命令レジスタ２４への入力の選択は、命令レジスタ２４の最上位の１ビットを用いて行ったが、状態制御レジスタ３１の１ビットを用いて行うようにすることもできる。この場合の構成図を図９に示す。

【０１６７】図７と異なる点は、命令レジスタ２４への入力データを選択するマルチプレクサ２３の制御を、状態制御レジスタ３１の１ビットを用いて行うことである。このような構成で、ＭＩＭＤ動作を行うための手順を次に示す。なお、図７の実施例と共通の処理は、記述を省略した。

【０１６８】１．ＳＩＭＤ処理

状態制御レジスタ３１の、マルチプレクサ２３を制御するビットを、制御プロセッサ１１からの命令を命令レジスタ２４に入力するように設定する。以後、状態制御レジスタ３１のマルチプレクサ２３を制御するビットを書き換えない限り、演算プロセッサ１２は制御プロセッサ１１からの命令にしたがって処理を行う。

【０１６９】２．ＭＩＭＤ処理への移行

制御プロセッサ１１の命令には、各演算プロセッサ１２のローカルメモリ１４に格納されている命令列に実行を移行するための命令が用意されている。この命令は、図８に示すような命令列の開始アドレスの格納されている番地を指定するフィールドを持つ。この命令が制御プロセッサ１１で実行されると、次のような手順で各演算プロセッサ１２の命令列に処理が移行される。

【０１７０】（１）指定されたアドレステーブルに格納されている命令列の開始アドレスがグローバルバス１５を介して全演算プロセッサ１２に放送される。

【０１７１】（２）制御プロセッサ１１からの命令による制御で、各演算プロセッサ１２で、（１）で放送されたＭＩＭＤ用命令列の開始アドレスが、プログラムカウンタ２５に格納される。

【０１７２】（３）制御プロセッサ１１からの命令による制御で、各演算プロセッサ１２で、プログラムカウンタ２５で示されるローカルメモリ１４の開始アドレスから、ＭＩＭＤ処理のための命令が読み出され、命令レジスタ２４に格納される。

【０１７３】（４）状態制御レジスタ３１の、マルチプレクサ２３を制御するビットを、命令レジスタ２４への入力データとして、ローカルメモリ１４から読み出したデータを選択するように設定する。

【０１７４】（５）命令レジスタ２４からの指示によって処理が行われる。この処理は、各演算プロセッサ１２で異なっていても良いため、ＭＩＭＤ処理となる。

【０１７５】３．ＭＩＭＤ方式による処理

前実施例と同様に、プログラムカウンタ２５には、ＭＩＭＤ用命令列の先頭アドレスが格納されており、この値をインクリメントさせながら、ローカルメモリ１４から命令列を順次読み出すことができる。状態制御レジスタの設定により、ローカルメモリ１４から読み出された命令列は、命令レジスタ２４に格納されるようにマルチプレクサ２３が制御される。これによって、ローカルメモリ１４に格納された命令列を順次読み出し、実行することができる。

【０１７６】４．ＭＩＭＤ方式からＳＩＭＤ方式への移行

ＭＩＭＤ処理の終了は、各演算プロセッサ１２でまちまちであるため、制御プロセッサ１１は、全演算プロセッサ１２でのＭＩＭＤ処理の終了を確認してからＳＩＭＤ処理を開始する必要がある。

【０１７７】（１）制御プロセッサ１１は、ＭＩＭＤ処理を起動した後、待ち状態に入る。この状態では、制御プロセッサ１１は、演算プロセッサ１２に対してＮＯＰ（No Operation）コードを発行する。

【０１７８】（２）各演算プロセッサ１２のＭＩＭＤ処理用命令列の最後尾に、状態制御レジスタ３１の同期ビ

ットをセットし、かつ、マルチプレクサ２３を制御するビットを制御プロセッサ１１からの命令を命令レジスタ２４に入力するように設定する命令を挿入しておく。

【０１７９】（３）各演算プロセッサ１２では、ＭＩＭＤ処理用命令列の最後のコードを実行すると、状態制御レジスタ３１の同期ビットがセットされ、さらに命令レジスタ２４への入力が、以後は制御プロセッサ１１が放送する命令を入力するようにマルチプレクサ２３が切り換わる。

【０１８０】（４）制御プロセッサ１１は、（１）で演算プロセッサ１２のＭＩＭＤ処理を起動した後、全演算プロセッサ１２で同期ビットに１がセットされたことを認識するまでＮＯＰ命令を発行し続ける。早く処理が終わった演算プロセッサ１２は、制御プロセッサ１１からＮＯＰコードを受け取って実行するが、副作用はない。（５）制御プロセッサ１１は、全演算プロセッサ１２の同期ビットに１がセットされたこと、すなわち全演算プロセッサ１２でＭＩＭＤ処理が終了したことを確認したら、ＮＯＰコードの発行を中止し、ＳＩＭＤ処理用の命令の発行を開始する。

【０１８１】（６）以降は、制御プロセッサ１１からの命令にしたがって、全演算プロセッサ１２でＳＩＭＤ処理が行われる。

【０１８２】このように、命令レジスタ２４への命令の入力の選択を、状態制御レジスタによって設定することもできる。この方法では、制御レジスタの設定のために命令が必要となるが、演算結果によって、制御プロセッサ１１からの命令列か、メモリ１３に格納されている命令列かのどちらかを実行するかを選択することも可能で、より柔軟性の高い処理が実現できる。

【０１８３】以上で述べた実施例では、制御プロセッサ１１は解読済みの命令、すなわちマイクロ命令を各演算プロセッサ１２に放送する方法を示した。これに対して、解読する前の機械語命令をそのまま放送することも可能である。この場合、各演算プロセッサ１２に、機械語を解読したマイクロ命令を発生するための機能を設ける必要があり、ハードウェア量は増加する。

【０１８４】しかし、近年開発されているＲＩＳＣ（Reduced Instruction Set Computer）プロセッサでは、機械語の解読は比較的簡単に行われるため、それほどハードウェア量は増加しないと考えられる。もちろん、このような場合でも、第２の発明で示したように、ＳＩＭＤ処理とＭＩＭＤ処理の両方を可能にする構成が可能である。

【０１８５】第３の発明

図１０は、第３の発明の並列計算機に係わる一実施例の構成を示す図である。同図には、簡単なモデルとして１次元リング結合の並列計算機を示した。この図に示す通り、１６個のＰＥ０～１５がリング状に配置され、各々のＰＥ０～１５は隣のＰＥと通信を行うものとする。

【０１８６】問題サイズが１６の倍数の時、分割されたタスクのＰＥへの割り付けは容易である。すなわち例えばサイズが１６の場合、ＰＥ０には１番目のタスクを割り当て、ＰＥ１には２番目のタスクの割り付けを行い、ＰＥ１５には１６番目のタスクを割り付ける。この場合、各ＰＥに均等にタスクが割り付けられ、バランス良く処理を実行できる。

【０１８７】しかし、問題サイズが１０の場合、１６台のＰＥ０～１５の内の１０台にタスクを割り付けるが、例えばＰＥ０からＰＥ９までに割り付けるとする。この時ＰＥ１０からＰＥ１５までは休止となるので、識別回路にＰＥ０からＰＥ９までが稼動であり、ＰＥ１０からＰＥ１５までが休止である情報をセットする。各ＰＥのバイパス回路は識別回路からの情報により通信時のバイパスを行う。

【０１８８】本例ではＰＥ１０からＰＥ１５のバイパス回路がバイパスを行う。これによりソフトから見たハードウェアの構成はＰＥ０からＰＥ９までがリング状に接続された１０台並列の１次元接続の並列計算機となる。左から順にＰＥ０、ＰＥ１…ＰＥ９と並んでいる場合、ＰＥ９の右の通信バスはＰＥ０の左側の通信バスに接続される。

【０１８９】図１１，１２にバイパス回路の詳解図を示す。図１１はＰＥ間が双方向バスで通信を行う場合のアドレス／データバスである。バイパス経路はＰＥを飛び越すように、ＰＥの右側バスと左側バスに接続する。双方向バスには双方向３－ｓｔａｔｅバッファ４１を挿入する。３－ステートバッファには識別回路４２から自ＰＥの稼動／休止情報が送られる。ＰＥが稼動の場合は３－ｓｔａｔｅとなり、休止の場合はバッファがオープンとなり、通信可能となる。

【０１９０】双方向バッファ４１の方向制御は並列計算機の仕様による。例えばＳＩＭＤ型並列計算機で、同期的に処理を実行する場合、データの転送方向は全てのＰＥで同じであるため、ＰＥの制御を行うＡＣＵ（Ａｒｒａｙ　Ｃｏｎｔｒｏｌ　Ｕｎｉｔ）から転送方向を指定するだけでよい。

【０１９１】一方、ＭＩＭＤ型並列計算機では、隣接通信は非同期である場合が多く、そのためハンドシェイクを行う必要がある。ハンドシェイクには要求信号と許可信号の信号線が必要となる。この場合、双方向バッファ４１の方向は、アクセス要求を出した側とアクセスが読み出しか書き込みかに依存してくる。そのため、バッファ４１の方向はバッファ４１に付属の制御回路で要求信号とＲｅａｄ／Ｗｒｉｔｅ信号から決定する。

【０１９２】ＭＩＭＤ型並列計算機で必要となる要求信号、許可信号、Ｒｅａｄ／Ｗｒｉｔｅ信号などは一方向の信号である場合が多い。特に今問題としているＰＥ間での転送は通信相手が固定できるためなおさらである。通信方向が１方向の場合の詳解図が図１２である。例え

ば右方向への要求信号を考えると、自ＰＥの左のＰＥから自ＰＥに向けられた要求信号と、自ＰＥから右のＰＥに送る要求信号を識別回路４２からの稼動／休止の信号により、セレクタ４３ａで選択して右のＰＥへ送る。

【０１９３】例えば自ＰＥが稼動の場合、右のＰＥに送る要求信号は自ＰＥからの要求信号をセレクタ４３ａで選択して送り、自ＰＥが休止している場合は左のＰＥから自ＰＥに送られてくる要求信号を右のＰＥに送る。これにより、自ＰＥが休止の場合は左のＰＥからの要求信号は直接右のＰＥに送られることになり、バイパスを行うことが出来る。

【０１９４】上記例ではＰＥ１０からＰＥ１５のかたまりを休止とした。しかし、このように局地的にかたまって休止にした場合、バイパスする距離が大きくなる。上記例では６ＰＥ分の距離をバイパスすることになる。図１１の場合、バイパスの際にはバッファ４１を通過する。連続してバイパスする数が増えればそれだけ通過するバッファ４１の数は増え、なおかつ物理的な距離も遠くなり、通信の為の時間が増加してしまう。

【０１９５】上記ＭＩＭＤ型並列計算機でのハンドシェイク方式ならば、多少通信時間に誤差が生じても防ぐことが出来るが、ＳＩＭＤ型並列計算機での同期方式では、各ＰＥ間での通信所要時間にばらつきがあると、問題が生ずる。

【０１９６】実際には、転送命令が４０ｎｓで実行しなければならず、隣接ＰＥ間で転送する場合２５ｎｓ必要な場合、バイパス１段を通るのに５ｎｓの遅延が生ずるとすると、１５÷５で最大３ＰＥ分のバイパスしか許されず、上記例での６個のＰＥのバイパスは不可能となる。

【０１９７】そのためバイパスするＰＥを選択する必要がある。例えば休止にするＰＥを１つおきにすれば、ＰＥ間には１ＰＥ分の遅延しか生じなくなる。上記例では休止するＰＥを例えばＰＥ５，ＰＥ７，ＰＥ９，ＰＥ１１，ＰＥ１３，ＰＥ１５と選択することにより可能となる。上記遅延の制約がある場合、上記１６台並列の計算機では、４台並列から、１６台並列まで自由に並列台数を変えることの出来る１次元リング結合並列計算機が実現できる。

【０１９８】また、上記例では各ＰＥ０～１５の保有するＰＥ番号（０～１５）を固定としているが、これだと休止するＰＥがある場合、稼動ＰＥの番号が連続とならない。実行するプログラムが連続したＰＥ番号を必要とする場合は、ＰＥ番号を、稼動するＰＥに与えるデータの一変数の値とすることで連続したＰＥ番号とすることができる。

【０１９９】第３の発明は仮想プロセッサの概念にも応用できる。すなわち、１６台並列のＰＥ０～１５に問題サイズが１６以上のタスクを割り付ける場合を考える。タスクの数が１６の倍数の場合、各ＰＥ０～１５に均等に割り付けることが出来る。例えば３２のタスクを割り付ける場合、各ＰＥ０～１５に２タスクずつ割り付ければ、通常の仮想プロセッサの処理で何等問題ない。

【０２００】すなわち各ＰＥ内部の２つのタスクを処理するとき、それぞれのタスクを処理するために２つの状態を示す状態保持手段を用い、タスク１とタスク２を各々実行するとき上記状態を切り替えることにより、現在どちらの処理を実行しているかを認識し、仮想プロセッサを実現し、１６台で３２のタスクを処理する。

【０２０１】しかし、例えばタスク数が２６の場合、各ＰＥ０～１５に均等に割り付けようとすると、タスクを１個受け持つＰＥと、２個受け持つＰＥとができる。各ＰＥが受け持つ２つのタスクをタスク１とタスク２とすると、タスク１を実行する場合は１６台のＰＥが一斉に行えば良いが、タスク２を実行する際は１０台のＰＥが稼動となり、残り６台のＰＥが休止となる。

【０２０２】つまりタスク２を実行する際は、上記例での１０個のタスクを実行する場合と同様の状態となる。そのため、タスク１を実行する場合は１６台全てが稼動し、タスク２を実行する場合はバイパス回路を使用して１０台並列の計算機に切り替えて実行を行えば仮想マシンとして２６台並列の並列計算機として使用できる。バイパス回路の使用／非使用は、上記状態保持手段に、上記識別回路４２を連動することにより実現できる。すなわちタスク１を処理する状態の時、識別回路４２は全てのＰＥ０～１５が稼動であるとしてバイパス回路を用いず、タスク２を処理する状態の時、識別回路４２は与えられているＰＥの使用／非使用の情報からＰＥの稼動／休止を決定し、バイパス回路の使用／非使用を決定する。

【０２０３】次に２次元格子トーラス結合の場合を示す。基本的には１次元リング結合と同様であり、バイパスの方向が列方向と行方向の２方向になる。このため、図１１，１２で示した双方向バッファ４１あるいはセレクタ４３も、列方向と行方向の２方向分必要となる。制御の簡単化を考えると、１次元リング結合で１ＰＥが休止の対象となったのに対し、２次元格子トーラス結合では行あるいは列が休止の対象となる。

【０２０４】図１３に８×８のアレイに問題サイズが６×５の処理を割り付けた場合を示す。まず行方向に関しては、８行のＰＥ列のうち、６行を稼動とする。各ＰＥを（Ｘ，Ｙ）で表すとすると、１次元リング結合の場合と同様に、局所的に休止ＰＥを固めないようにすると、ＰＥ（０，Ｙ）、ＰＥ（１，Ｙ）、ＰＥ（２，Ｙ）、ＰＥ（３，Ｙ）、ＰＥ（４，Ｙ）、ＰＥ（６，Ｙ）の６行がそれぞれ稼動となる（図中、丸印）。

【０２０５】列方向に関してはＰＥ（Ｘ，０）、ＰＥ（Ｘ，１）、ＰＥ（Ｘ，２）、ＰＥ（Ｘ，４）、ＰＥ（Ｘ，６）の５列がそれぞれ稼動となる。休止となったＰＥ列のＰＥ（５，Ｙ）、ＰＥ（７，Ｙ）（図中、×

印）は行方向のバイパスを行い、ＰＥ（Ｘ，３）、ＰＥ（Ｘ，５）、ＰＥ（Ｘ，７）はそれぞれ列方向のバイパスを行う。

【０２０６】特に休止の行と列の交差にあるＰＥ（３，５）、ＰＥ（３，７）等は縦方向と横方向の双方をバイパスする。２次元格子トーラス結合において、仮想プロセッサを実現する際も、１次元リング結合で挙げた例と同様にして行うことが出来る。

【０２０７】３次元格子トーラス結合の場合は、休止の対象が２次元の行／列に対し、面が対象となる。

【０２０８】上記バイパス回路はＰＥと同一チップに納めたときに有効となる。すなわち１方向に複数のチップをバイパスする場合、バイパス経路がチップ外にある場合よりもはるかに高速にバイパスを行うことが出来る。特に複数ＰＥ／ｃｈｉｐの場合、さらにバイパスの高速化を行うことが出来る。

【０２０９】次にメモリアクセスに関して説明する。例としてまず１６台並列の１次元リング結合の並列計算機を挙げる。上記例で１６台並列の１次元リング結合型計算機で問題サイズ１０の処理を行う場合を挙げたが、この時上記並列計算機が分散メモリ型であり、各ＰＥ０～１５がローカルメモリ（ＬＭ）を持つ場合、休止となっているＰＥのローカルメモリも使用されないことになる。

【０２１０】例えば問題サイズは４だが、１つのタスクが必要とするメモリ量が１ＰＥの持つメモリの数倍の容量を必要とする場合もある。通常この場合は１つのタスクのデータを複数に分け、おおもとのデータ保持部から随時転送する方法をとる。しかしこの場合、データを分割したためにデータが分割されていることを意識したプログラムが必要となる。

【０２１１】そこで、休止しているＰＥがある場合、同ＰＥのローカルメモリを、稼動しているＰＥが使用できるようにすることにより、１ＰＥあたりのメモリ容量を増やすことができる。

【０２１２】図１４に１６台並列１次元リング結合の並列計算機の例を示す。同例で、上記バイパスに関する実施例を条件とすると、本並列計算機は４台から１６台までのスケーラブル並列計算機となる。

【０２１３】今、問題サイズが８とすると、稼動するＰＥはＰＥ０，２，４，６，８，１０，１２，１４の８台となる。各ＰＥが自ＰＥの右側のローカルメモリ（ＬＭ）４４を切り換えスイッチ４５で直接アクセス出来るようにすると、８台並列で、各ＰＥのローカルメモリ４４の容量が２倍となる。

【０２１４】問題サイズが５の場合は、稼動ＰＥをＰＥ０，３，６，９，１２とし、ＰＥ０はＰＥ１，２のローカルメモリ４４を、ＰＥ３はＰＥ４，５のローカルメモリ４４を使用することにより、３倍のメモリ容量となる。４台のＰＥが稼動の場合は４倍のメモリ容量となる。アクセスのためのアドレス空間は、キャッシュメモリで用いるようなバンク方式あるいはウィンドウ方式も考えられるが、連続したアドレス空間として使用しても良い。

【０２１５】ここで、どのＰＥを休止とし、どの稼動ＰＥがどの休止ＰＥのローカルメモリ４４を使用できる用にするかを決定することが問題となる。まず稼動ＰＥをＰＥ０，１，２，３，４，５，６，７とし、ＰＥ８～１５を休止として、ＰＥ０がＰＥ８のメモリを、ＰＥ７がＰＥ１５のメモリを使用することにしても良いが、まず稼動するＰＥは上記バイパスにおける制約条件もあるが、各ＰＥが番号順に左から順に実装されている場合、ＰＥ０からＰＥ８への物理的距離が遠くなる。そのためアクセス対象は実装時のモジュール間の距離を考慮して決定すべきである。

【０２１６】上記例では、割り付けられるタスクの大きさが均等の場合を念頭に述べたが、タスクの大きさすなわち必要となるメモリ量が異なっても良い。すなわち今６個のタスクがあり、それぞれ必要となるメモリ量が４，２，３，４，１，２であった場合、稼動ＰＥはＰＥ０，４，６，９，１３，１４となり、ＰＥ０はＰＥ１，２，３をＰＥ４はＰＥ５を、ＰＥ６はＰＥ７，８を、ＰＥ９はＰＥ１０，１１，１２を、ＰＥ１４はＰＥ１５のローカルメモリ４４をそれぞれ使用する。

【０２１７】このように、単純にタスクを左から割り付けるとすると、タスクの大きさに適応するためには、前述した通信時間の関係から、全てのＰＥが他の少なくとも３つのＰＥにアクセスする経路を設けなければならず、回路／配線の増大を招く。これを解消するため、例えばＰＥ０，４，８，１２の各ＰＥは自ＰＥの右隣３ＰＥへの経路をもち、他の偶数番号ＰＥは隣のＰＥへの経路を持つように限定し、大きさの異なるタスクの割り付けはスケジューリングを行い対処する。すなわちＰＥをグループ化する。

【０２１８】これは複数ＰＥを１つのチップに載せた場合にも有利となる。例えば、４ＰＥを１つのチップに配置し、上記切り換えスイッチ４５などのアクセス回路もチップ内に納める。これにより、アクセス回路の切り換えによる速度低下を防ぐことが出来る。同一チップ上ならば、多少の回路／配線が増加しても速度の低下は防ぐことが出来る。そのため、同一チップに配置したＰＥ内では、全てのＰＥが他のＰＥのローカルメモリ４４を使用できるようにしても良い。こうすることにより、大きさの異なるタスクへの対処の柔軟性は上がる。

【０２１９】次に２次元格子トーラス結合のメモリアクセスの場合を示す。２次元の場合、休止の対象は行／列である。そのため、アクセス回路も行方向と列方向に設ける。上記１次元リング結合で、チップに載せることも考慮して、ＰＥをグルーピングしたが、２次元の場合でも同様のことがいえる。

【０２２０】図１５に２×２のグループを考える。ＰＥ（０，０）は他のＰＥのローカルメモリ４４が使用できる。ＰＥ（１，０）とＰＥ（０，１）はＰＥ（１，１）のローカルメモリ４４が使用できるようにする。これにより列方向、行方向のスケーリングを利用してメモリ容量の増加を行うことが出来る。また、ＰＥ（１，０）とＰＥ（０，１）は休止するため、バスパス回路４６を設けている。

【０２２１】３次元格子結合に関しても上記例と同様にすることにより、メモリ容量の増加を行うことが出来る。

【０２２２】次にハイパーキューブでのメモリアクセスについて説明する。並列計算機の静的結合方式の１つであるハイパーキューブは、通信距離の半径が小さいなどの利点が挙げられるが、スケーラビリティの点でも有利である。すなわちハイパーキューブの次元を１つ下げるには、余分な次元に対応する通信経路を使用しなければよく、特別な回路は必要ない。

【０２２３】例として４次元から３次元にする例を図１６に示す。稼動するＰＥをＰＥ０～７とし、稼動しないＰＥをＰＥ８～１５とすると、容易に１次元下がったハイパーキューブ結合の並列計算機となる。今、ＰＥ８～１５が休止であるので上記メモリアクセス路の例で述べたように、休止のＰＥのローカルメモリ４４を、稼動するＰＥのメモリとして使用することが出来る。

【０２２４】すなわちＰＥ８のローカルメモリ４４をＰＥ０が使用し、ＰＥ９のローカルメモリ４４をＰＥ１が使用する。通信経路０－８，１－９，２－１０，３－１１，４－１２，５－１３，６－１４，７－１５を使用しないので、メモリアクセス用パスとして上記通信経路を使用しても良い。

【０２２５】次に複数ＰＥを１つのチップに配置する場合の配線について説明する。図１７に２×２の４つのＰＥ４７ａ，ｂを１つのチップ４８に載せた場合の例を示す。ＰＥ４７のモジュールをチップ４８内に配置する場合、各ＰＥ間の通常の通信路の配線が場所を取る。そこで、アルミ等の配線層を複数化する近年のチップ製造技術を用い、休止ＰＥ４７ａをバイパスする経路を休止ＰＥ４７ａの上を通過するように配線する。これにより、配線領域を縮小することが出来、余分な配線の引き回しによる遅延の増大を防ぐことが出来る。

【０２２６】同様に、休止ＰＥ４７ａのローカルメモリ４４へのアクセス経路もＰＥ４７ａの上を通過するように配線することにより、配線領域の縮小が行える。

【０２２７】第４の発明

図１８は、第４の発明の並列計算機で用いられる演算プロセッサの実施例である。

【０２２８】ルータ部５１と演算処理部５２から構成される。ルータ部５１はフリットを蓄えるバッファ５３ｘ～５３ｐと、フリットの出力先を決定するスイッチ５４から構成される。

【０２２９】Ｐｉｎ，Ｐｏｕｔはルータ部５１と演算処理部５２とを接続する通信チャネルである。Ｘｉｎ，Ｘｏｕｔ，Ｙｉｎ，Ｙｏｕｔはそれぞれ隣接する演算プロセッサとの間の入力および出力のための通信チャネルである。上記の演算プロセッサを用いて、例えば図１９に示すような並列計算機が構築される。各演算プロセッサ５５は、互いに通信チャネル５６で接続される。

【０２３０】図２０は本実施例で通信されるメッセージのフォーマットの一例である。一つのメッセージは複数のフリットに分割される。先頭の二つのフリットは図２０（ａ）のフォーマットをしており、図中ａｄｄｒｅｓｓで表されるメッセージの宛先が書かれている。左右方向の宛先が書かれたフリットに上下方向の宛先が書かれたフリットが続く。各方向の宛先は送信側プロセッサと受信側プロセッサとの間の相対距離で表される。

【０２３１】３つめのフリットからメッセージの本体となる。そのフォーマットは、図２０（ｂ）および（ｃ）に示すとおりである。フリットは図中ｄａｔａで表されるメッセージの格納されている部分と、図中右端のメッセージの継続あるいは終了を表す終了ビット（ｅｎｄ＿ｂｉｔ）からなる。メッセージが継続する場合は図２０（ｂ）のフォーマットのようにｅｎｔ＿ｂｉｔは０であり、メッセージが終了する場合は図３（ｃ）のフォーマットのようにｅｎｄ＿ｂｉｔは１となる。

【０２３２】図２１は本実施例のルータ部５１で行われるルーティング方法である。まず左右方向の宛先が一致するまで左右方向に隣接する演算プロセッサ間で通信を行う（ステップ１００，１０１）。続いて上下方向の宛先が一致するまで上下方向に隣接する演算プロセッサ間で通信を行う（ステップ１０２，１０３）。第４の発明はこのルーティング方法に限定されず、いかなるアルゴリズムにも適応可能である。

【０２３３】図２２はスイッチ５４の内部構成例である。図１９における左右方向のスイッチ５７と上下方向のスイッチ５８とからなる。左右方向のスイッチ５７と上下方向のスイッチ５８は同じ構成で実現される。それを図２３に示す。二つの出力先決定回路６１と、二入力二出力のクロスバスイッチ６２から構成される。クロスバスイッチ６２には複数の方向から入力が与えられるので、これらが衝突する場合は調停が行われる。

【０２３４】図２４は出力先決定回路６１の構成例である。入力されたフリットのａｄｄｒｅｓｓ部が宛先比較部６３で比較される。内部状態保存部６４に記憶されている現在の状態ｓｔａｔｅで、先頭フリットか否かが判定される。Ｘ方向あるいはＹ方向における宛先が自演算プロセッサと同じメッセージであれば信号ｅｑｌを、そうでなければ信号ｎｅｑを出力先決定部６５に出力する。出力先決定部６５では、これらの制御信号とフリットのｅｎｄ＿ｂｉｔ，そして内部状態ｓｔａｔｅから出

力先制御信号を発生する。

【０２３５】ｃｔｌ＿ｉｎは自プロセッサ（スイッチ５８の場合）あるいは次のスイッチ５８（スイッチ５７の場合）を、ｃｔｌ＿ｏｕｔは隣接するプロセッサを出力先に選択するための制御信号である。ｃｔｌ＿ｉｎ、ｃｔｌ＿ｏｕｔはいずれもクロスバスイッチ６２に入力される。

【０２３６】出力先決定部６５は次状態を決定し、内部状態保存部６４に記憶する。さらに出力先決定部６５はデクリメンタ６６への制御信号ｃｔｌ＿ｒｅｍｏｖｅ、ｃｔｌ＿ｄｅｃも発生する。ｃｔｌ＿ｒｅｍｏｖｅはメッセージから先頭のフリットを取り除くための制御信号で、ｃｔｌ＿ｄｅｃはａｄｄｒｅｓｓを１減ずるための信号である。

【０２３７】デクリメンタ６６はこれらの制御信号にしたがって、宛先が自演算プロセッサと同じ場合は先頭のフリットを取り除き、そうでないときは先頭のフリットに書かれたａｄｄｒｅｓｓを１減ずる。先頭フリットでない場合は、何も操作は行わないでフリットを通過させる。デクリメンタ６６はまた、リクエスト信号ｒｅｑを発生する。

【０２３８】図２５は内部状態保存部６４の状態遷移の様子を表している。初期状態Ｓ０にいる時にフリットを受け取った場合には、このフリットは先頭であるので、ａｄｄｒｅｓｓにしたがってＳ１あるいはＳ２に遷移する。ａｄｄｒｅｓｓが自演算プロセッサと同じ場合にはＳ１に遷移する。メッセージが続いている場合にはＳ１あるいはＳ２の状態を継続し、ｅｎｄ＿ｂｉｔによりメッセージの終了を検出すると初期状態Ｓ０に戻る。

【０２３９】第４の発明を用いた場合に、従来の問題が解決することを図２６を用いて説明する。図２６において、５５ａ～５５ｅは演算プロセッサであり、それぞれバッファ５３ｘ，５３ｙ，５３ｐ、スイッチ５４および演算処理部５２から構成される。

【０２４０】出力先の演算プロセッサ５５ｂのバッファが一杯で通信が停止しているフリット６７が演算プロセッサ５５ａのバッファ５３ｘに蓄えられているとする。このとき演算プロセッサ５５ｃから別のメッセージのフリット６８が入力されるとする。

【０２４１】このメッセージの出力先が演算プロセッサ５５ｅとすると、演算プロセッサ５５ｅのバッファに空きがあれば、フリット６８は出力されるべきである。フリット６８はバッファ５３ｙに入力されるので、バッファ５３ｘには関係なく演算プロセッサ５５ｅに出力することができる。

【０２４２】すなわち、出力先の通信チャネルが空いてさえいれば、通信の停止している他のメッセージに妨害されることなく、出力可能である。したがって、並列計算機全体としての通信時間が短縮し、スループットも向上する。

【０２４３】従来例の場合では、先にバッファに蓄えられているフリットの通信が停止してしまうと、それ以降にバッファに蓄えられたフリットは、たとえ出力先のチャネルが空いていても出力されなかった。しかし第４の発明では、受信されたフリットは、先に出力先を決定しバッファに蓄えられる。したがって、ブロックされている通信チャネルのメッセージだけが通信を停止し、他のメッセージは通信を妨害されることはない。

【０２４４】本実施例では二次元のネットワークの場合を説明したが、ルータ部５１にスイッチ５７を増設するだけで高次元のネットワークに拡張可能である。たとえば三次元の場合は図２７のようにスイッチ３９を増設すればよい。また通信ネットワークは、本実施例のようなトーラス構造に限らず、ハイパーキューブ構造など、他のいかなる構造にも適応可能である。

【０２４５】さらに第４の発明は、ワームホール・ルーティングに限らず、ストア・アンド・フォワード・ルーティングなど、他のいかなる通信方式にも適応可能である。

【０２４６】第５の発明

図２８は第５の発明の並列計算機で用いられるプロセッサの構成図である。Ｘｉｎ、Ｘｏｕｔはそれぞれ隣接するプロセッサとの間の入力および出力チャネルである。Ｐｉｎ、Ｐｏｕｔはルータ部７１と演算処理部７２を接続する通信チャネルである。ルータ部７１はＸｉｎ、Ｐｉｎから受信したメッセージの転送先を決定し、Ｘｏｕｔあるいは Ｐｏｕｔに出力するものである。バイパススイッチ７３は、第５の発明のために追加された構成であり、Ｘｉｎからの入力をルータ部７１をバイパスして、直接Ｘｏｕｔに出力するブロックである。

【０２４７】図２９は、図２８のプロセッサで構成される並列計算機である。各プロセッサ７４ａ～７４ｅが通信チャネル７５によって接続されている。図示していないが、リクエスト信号とアクノリッジ信号も接続されている。

【０２４８】本実施例で通信されるメッセージのフォーマットは、図２０で示したものと同様である。

【０２４９】図３０はルータ部７１の内部構成例である。プロセッサはＸｉｎ方向のプロセッサからリクエスト信号を受けると、ラッチ７６にフリットを受け取る。このフリットがメッセージの先頭であれば、転送先決定ブロック７７で自プロセッサに取り込むか、バイパスするか、あるいは隣接するプロセッサに転送するかを決定する。転送先決定ブロック７７は同時に出力先へのリクエスト信号を生成する。クロスバスイッチ７８にはメッセージとリクエスト信号、そして転送先決定ブロック７７からメッセージの出力先を示す信号が与えられる。この信号は２方向から与えられるので、これらが衝突した場合には調停を行って、メッセージとリクエスト信号を出力先へスイッチする。その結果、フリットはバッファ

７９に入れられる。同時に転送先決定ブロック７７の発したリクエスト信号は出力先のプロセッサに与えられる。

【０２５０】図３１は転送先決定ブロック７７を構成するブロック図である。入力されたフリットが、メッセージの先頭である場合は宛先比較ブロック８１で宛先を比較する。先頭フリットであるか否かは、状態記憶ブロック８２に記憶されている現在の状態ｓｔａｔｅで判定される。宛先はプロセッサ間の相対距離で表されるので、宛先が０ならば自プロセッサ宛、０以外ならば他のプロセッサ宛である。

【０２５１】自プロセッサ宛ならば信号ｅｑｌを、他のプロセッサ宛ならば信号ｎｅｑを、出力先決定ブロック８３に出力する。出力先決定ブロック８３はこれらの制御信号と内部ステートにしたがって出力先制御信号を発生する。

【０２５２】ｃｔｌ＿ｂｙｐａｓｓはバイパスするための制御信号で、ｃｔｌ＿ｏｕｔ１、ｃｔｌ＿ｏｕｔ２はそれぞれ、バイパスしない場合の出力先としてＸｏｕｔ、Ｐｏｕｔを選択するための制御信号である。

【０２５３】ｃｔｌ＿ｂｙｐａｓｓは第５の発明において特徴となる制御信号であり、バイパススイッチ７３に入力されて、バイパス動作を行う。ｃｔｌ＿ｏｕｔ１、ｃｔｌ＿ｏｕｔ２はいずれもクロスバスイッチ７８に入力されて、出力先を要求する。また出力先決定ブロック８３は次の状態ｎｅｘｔ＿ｓｔａｔｅを決定し、状態記憶ブロック８２に記憶する。さらに出力先決定ブロック８３はデクリメンタ１２４に対する制御信号ｃｔｌ＿ｒｅｍｏｖｅ、ｃｔｌ＿ｄｅｃも発生する。

【０２５４】ｃｒｌ＿ｒｅｍｏｖｅは、デクリメンタ８４に入力される宛先を取り除くための制御信号で、ｃｔｌ＿ｄｅｃは宛先を１減ずるため制御信号である。デクリメンタ８４はこれらの制御信号にしたがって、宛先が自プロセッサであるときには宛先を取り除き、そうでないときには宛先を１減ずる。先頭フリットでない場合は、何もしないでフリットを通過させる。デクリメンタ８４はまたリクエスト信号ｒｅｑを発生する。

【０２５５】図３２は状態記憶ブロック８２の状態遷移を表す図である。初期状態Ｓ０のときにフリットを受け取ると、このフリットはメッセージの先頭であるので、宛先にしたがってＳ１、Ｓ２あるいは Ｓ３に状態を遷移する。例えば、メッセージをバイパスする場合はＳ１に、Ｘｏｕｔに出力する場合はＳ２に、Ｐｏｕｔに出力する場合はＳ３に遷移する。

【０２５６】メッセージが続いている間はＳ１～Ｓ３の状態に留まり、ｅｎｄ＿ｂｉｔによりメッセージの終了を検出すると初期状態Ｓ０に戻る。Ｓ１からＳ０に状態が遷移するとバイパスは解除される。すなわちバイパスしている状態でも、メッセージはルータ部７１に取り込まれている。ただし、取り込まれるメッセージは終了を検出する目的だけに用いられ、中継されたりはしない。

【０２５７】第５の発明のバイパス動作を以下に説明する。図３３は図２８で示したプロセッサを用いて構成した並列計算機である。８５ａから８５ｅはメッセージを通信するためのチャネル（図中、ｄａｔａ）、８６ａから８６ｅはリクエスト信号、８７ａから８７ｅはアクノリッジ信号である。本実施例ではメッセージは単方向に通信されるが、双方向通信の場合も本質的な違いはない。各プロセッサは図３３（ａ）に示すように隣接するプロセッサ７４とチャネルおよび信号で接続される。

【０２５８】本実施例で、プロセッサ７４ａからプロセッサ７４ｅに中継プロセッサを全てバイパスしてメッセージを送信する場合を説明する。プロセッサ７４ａはプロセッサ７４ｂへリクエスト信号８６ａを発し、宛先を持ったフリットを７４ｂに送信する。プロセッサ７４ｂはリクエスト信号を受け取るとフリットを受信し、フリットに書かれている宛先を読みだす。

【０２５９】プロセッサ７４ｂは宛先ではないので、次の転送先プロセッサ７４ｃを選択する。同時にプロセッサ７４ｂは通信チャネル８５ａと８５ｂ、リクエスト信号８６ａと８６ｂ、およびアクノリッジ信号８７ａと８７ｂをバイパスする。リクエスト信号８６ａと８６ｂがバイパスされることにより、プロセッサ７４ｃにリクエスト信号が要求されることになる。

【０２６０】同様な操作がプロセッサ７４ｃ及びプロセッサ７４ｄで実行され、図３３（ｂ）のように経路が確定される。プロセッサ７４ｅはリクエスト信号を受けると、フリットを受信して宛先を読む。プロセッサ７４ｅは受信側プロセッサであるので、プロセッサ７４ｄに対してアクノリッジ信号を返す。このときプロセッサ７４ｂからプロセッサ７４ｄの内部でアクノリッジ信号８７ａから８７ｄはバイパスされているので、図３３（ｃ）のようにアクノリッジ信号は直ちにプロセッサ７４ａに到達する。

【０２６１】こうしてプロセッサ７４ａとプロセッサ７４ｅとの間のハンドシェイクが完成し、メッセージを転送する経路が確立される。以後は図３３（ｄ）のようにプロセッサ７４ａとプロセッサ７４ｅの間で２つめ以降のフリットの通信が行われる。この間プロセッサ７４ｂ～７４ｄは、終了検出を行うためにバイパスしているフリットを監視しつづける。そしてメッセージの終了を検出すると初期状態に戻る。

【０２６２】この方法を用いた場合に転送に必要な時間は、経路を確立するためのリクエスト信号の伝達にＤ、アクノリッジ信号の伝達に１、経路確立後のメッセージの送信に２×（Ｌ－１）必要で、全体で（Ｄ＋２×Ｌ－１）となる。このようにプロセッサ間の通信時間が短縮される。Ｄ＝４、Ｌ＝４の場合のタイミング図を図３４に示す。

【０２６３】以上はプロセッサ間でのメッセージの通信

をハンドシェイクで行った場合であるが、さらに通信を高速にするために、図３３（ｂ），（ｃ）の処理によって経路を確立せさたのち、図３３（ｅ）のようにプロセッサ７４ａとプロセッサ７４ｅをクロック同期させて通信を行う方法が考えられる。この場合、メッセージの送信に必要な時間は（Ｌ－１）となって、全体では（Ｄ＋Ｌ）となる。これは従来のワームホール・ルーティングのほぼ１／２の時間で、プロセッサ間の通信にかかる時間は大いに短縮される。Ｄ＝４、Ｌ＝４の場合のタイミング図を図３５に示す。

【０２６４】同期通信の場合は、プロセッサ間の距離が非常に大きいと、１クロックではフリットが到達できない可能性がある。このような場合を考慮して、通信を行っているプロセッサの間に存在するプロセッサの一部でメッセージを中継することを考える。すなわち、上述した実施例では、自プロセッサ宛でないメッセージは全てバイパスしていたが、転送先への距離に応じてバイパスと中継を分けるように、宛先比較ブロック８１および出力先決定クロック８３を変更する。

【０２６５】本実施例を用いて、例えば１クロックで到達可能な距離を８とした場合を考える。先頭のフリットに書かれているデータは、受信側プロセッサまでの相対距離であるので、この宛先が８の倍数であるときには中継し、そうでないときにはバイパスするように、宛先比較ブロック８１と出力先決定ブロック８３を変更する。

【０２６６】こうして１クロックで到達できる距離毎に中継プロセッサが選択される。このような変更で、非常に規模の大きなネットワークでも、従来例よりも高速にプロセッサ間通信を行うことができる。

【０２６７】なお、本実施例では一次元の場合を説明したが、第５の発明は一次元に限らず、容易に多次元に拡張可能である。図３６は二次元の場合のプロセッサ構成図である。Ｘｉｎ、Ｙｉｎ、Ｐｉｎからの入力を、Ｘｏｕｔ、Ｙｏｕｔ、Ｐｏｕｔに出力する。バイパススイッチ８８はバイパススイッチ７３に比べてバイパスする信号が一つ増えたブロックである。

【０２６８】このように次元を拡張するには、次元数Ｎ分だけルータ部７１とバイパススイッチ７３を用意する。これで各次元でバイパスが可能になるが、異なる次元間でもバイパス可能にするためには、バイパススイッチ８８のように他の次元からのバイパスも選択できるように、スイッチの入力を増やす。

【０２６９】

【発明の効果】以上のように、第１の発明の並列計算機は、並列度がプロセッサの数よりも大きい場合は通常のＳＩＭＤ型並列計算機と同様に実行することができ、並列度がプロセッサの数よりも少ない場合には、複数命令の並列実行を行なうことにより、実行時間の短縮とプロセッサの有効利用を図ることができる。また、通常のＳＩＭＤ型並列計算機とオブジェクト互換をもつようにすることが可能である。

【０２７０】また、第２の発明の並列計算機によれば、ＳＩＭＤ型並列計算機にわずかなハードウェアを付加するだけで、ＭＩＭＤ処理が可能になり、より柔軟性の高い処理が可能になる。

【０２７１】第３の発明の並列計算機によれば、処理する問題サイズに応じて使用しないＰＥをバイパスする事により、結合形態を維持したまま、問題サイズに最適な構成を取ることが出来る。使用しないＰＥのメモリを、使用するＰＥがアクセスできるようにすることにより、各ＰＥのメモリ容量を増やすことが出来る。

【０２７２】さらに、同一チップ内において、バイパス経路、メモリアクセス経路をＰＥの上を通過するように配線することにより、配線領域の縮小、遅延増大の防止を行うことが出来る。

【０２７３】第４の発明の並列計算機によれば、フリットは出力先毎に出力側のバッファに蓄えられるので、通信可能なメッセージは、通信が停止している他のメッセージによって妨害されることはない。その結果、通信時間が短縮され、スループットが向上する。

【０２７４】さらに、第５の発明の並列計算機によれば、中継プロセッサをバイパスすることにより、通信に必要な時間を大いに短縮することができる。このため、特にネットワークの規模が非常に大きくプロセッサ間の距離が大きな場合に有効である。

【図面の簡単な説明】

【図１】第１の発明における並列計算機の一実施例の構成図。

【図２】第１の発明における実施例で用いた処理の一例。

【図３】並列度が大きい場合の図２の処理を行なうために供給される命令列。

【図４】並列度が小さい場合の図２の処理を行なうために供給される命令列。

【図５】第１の発明の図１と異なる実施例の構成図。

【図６】第２の発明の並列計算機に係わる一実施例の構成を示すブロック図。

【図７】第２の発明の中心となる演算プロセッサの構成例を示す図。

【図８】図７で示したローカルメモリ中に作成されるアドレステーブルのメモリマップ図。

【図９】図７と異なる演算プロセッサの構成例を示す図。

【図１０】第３の発明における１次元リング結合並列計算機のバイパスに関する一例の構成図。

【図１１】第３の発明においてバイパス回路として用いた双方向バスの詳解図。

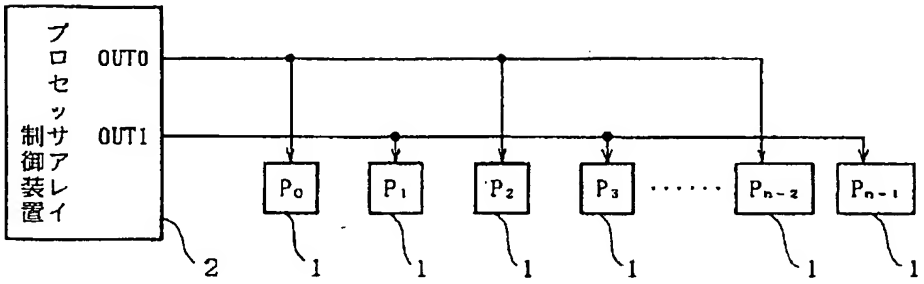
【図１２】第３の発明においてバイパス回路として用いたハンドシェイク用信号線の詳解図。

【図１３】第３の発明における２次元トーラス結合並列

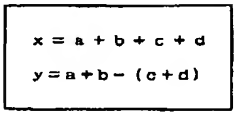
計算機のバイパスに関する一例の構成図。
【図１４】第３の発明における１次元リング結合並列計算機のメモリアクセスに関する一例の構成図。
【図１５】第３の発明における２次元トーラス結合並列計算機のバイパス及びメモリアクセスに関する一例の構成図。
【図１６】第３の発明におけるハイパーキューブ結合並列計算機のメモリアクセスに関する一例の構成図。
【図１７】第３の発明におけるバイパス経路及びアクセス経路のチップ上での配線例。
【図１８】第４の発明における演算プロセッサの一実施例を表す構成図。
【図１９】図１８で示した演算プロセッサで構成される並列計算機の構成図。
【図２０】第４の発明において通信されるメッセージのフォーマット例。
【図２１】第４の発明におけるルーティング方法を表すフローチャート。
【図２２】図１８で示したスイッチの内部構成図。
【図２３】図２２で示した一方向のスイッチの内部構成図。
【図２４】図２３で示した出力先決定回路の内部構成図。
【図２５】図２３で示した出力先決定回路の状態遷移図。
【図２６】図１８で示した演算プロセッサを用いた通信方法を表す図。
【図２７】第４の発明における３次元ネットワークの場合の一実施例を示す構成図。
【図２８】第５の発明におけるプロセッサの一実施例の構成を示す構成図。
【図２９】図２８で示したプロセッサを用いた並列計算機の構成図。
【図３０】図２８で示したルータ部の一例を示す図。
【図３１】図３０で示した転送先決定ブロックの構成図。
【図３２】図３０で示した転送先決定ブロックの状態遷移図。
【図３３】第５の発明におけるバイパス動作を表す図。
【図３４】第５の発明におけるバイパス動作時の転送時間を表すタイミング図。
【図３５】第５の発明におけるクロック同期による転送時間を表すタイミング図。
【図３６】第５の発明における２次元の場合のプロセッサ構成図。
【図３７】第２の発明に対する従来の演算プロセッサの構成を示す図。
【図３８】第４の発明に対する従来の演算プロセッサの構成図。
【図３９】図３８で示した演算プロセッサを用いた従来の通信方法を表す図。
【図４０】第５の発明に対する従来のプロセッサ構成図。
【図４１】第５の発明に対する従来のバイパス動作時の転送時間を表すタイミング図。
【図４２】第５の発明に対する従来のクロック同期による転送時間を表すタイミング図。
【符号の説明】
１　プロセッサ
２　プロセッサアレイ制御装置
３　セレクタ
ＯＵＴ０，１　出力端子
１１　制御プロセッサ
１２　演算プロセッサ
１３　全体メモリ
１４　ローカルメモリ
１５　グローバルバス
１６　ネットワーク
１７　同期バス
１８　フロントエンド計算機
２１　メモリアドレスバス
２２　メモリデータバス
２３　マルチプレクサ
２４　命令レジスタ
２５　プログラムカウンタ
２６　アドレスレジスタ
２７　データレジスタ
２８　内部バス
２９　ＡＬＵ
３０　レジスタファイル
３１　状態制御レジスタ
３２　プロセッサ番号レジスタ
３３　通信制御部
４１　双方向バッファ
４２　識別回路
４３　セレクタ
４４　ローカルメモリ
４５　切り換えスイッチ
４６　バスパス回路
４７　ＰＥ（演算要素）
４８　チップ
５１　実施例のルータ部
５２　演算処理部
５３ｘ，５３ｙ，５３ｐ　バッファ
５４　スイッチ
５５ａ～５５ｅ　演算プロセッサ
５６　通信チャネル
５７，５８，６９　一方向のスイッチ
６１　出力先決定回路
６２　クロスバスイッチ

６３　出力先選択部
６４　内部状態保存部
６５　出力先決定部
６６　デクリメンタ
７１　ルータ部
７２　演算処理部
７３，８８　バイパススイッチ
７４ａ～７４ｅ　プロセッサ
７５，８５ａ～８５ｅ　通信チャネル
７６　ラッチ
７７　転送先決定ブロック
７８　クロスバスイッチ
７９　バッファ
８１　宛先比較ブロック
８２　状態記憶ブロック
８３　出力先決定ブロック
８４　デクリメンタ
８６ａ～８６ｅ　リクエスト信号
８７ａ～８７ｅ　アクノリッジ信号

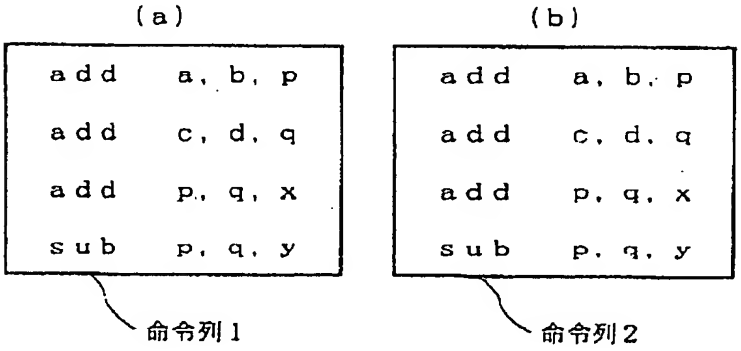
【図１】



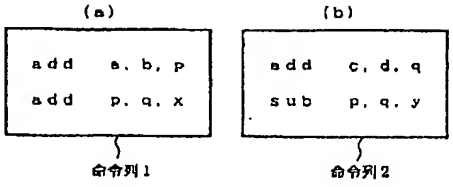
【図２】



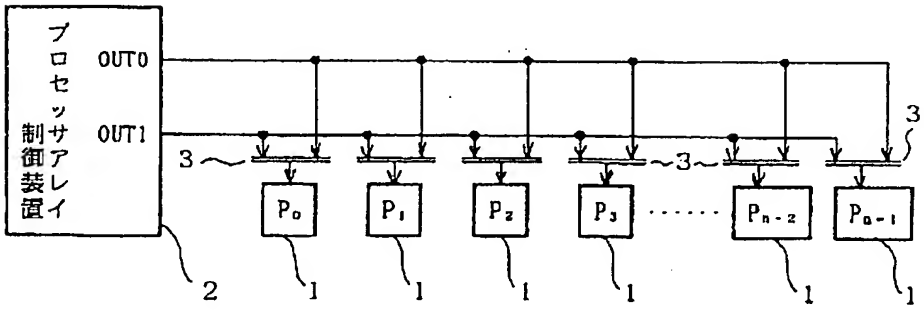
【図３】



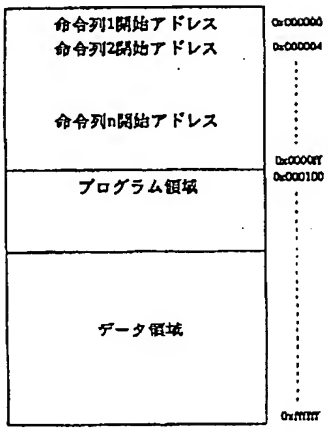
【図４】



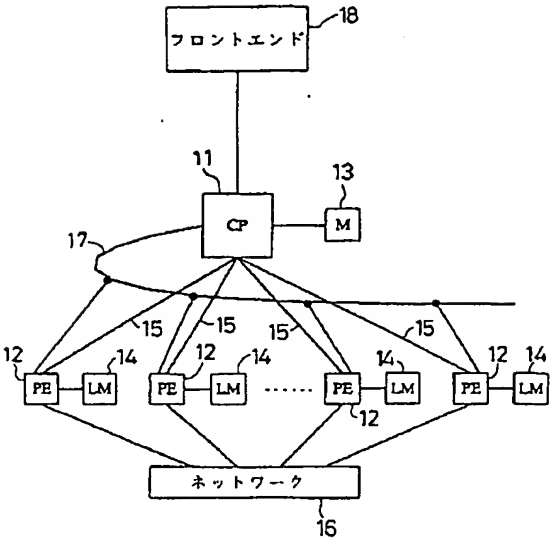
【図５】



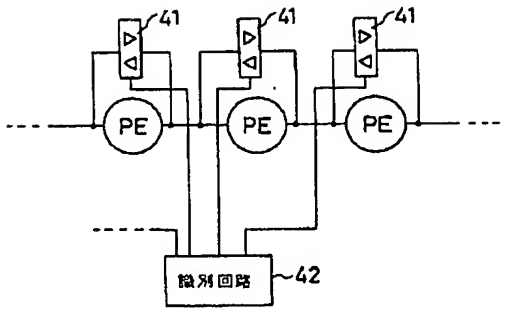
【図８】



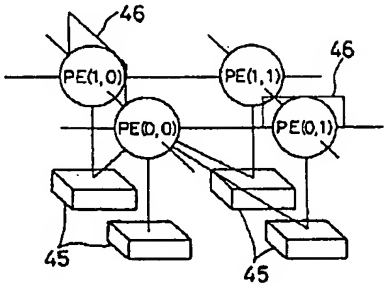
【図６】



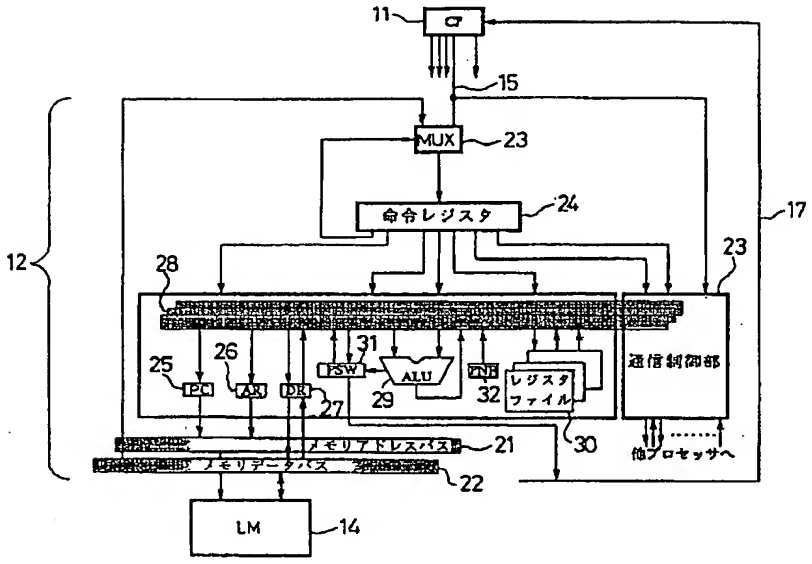
【図１１】



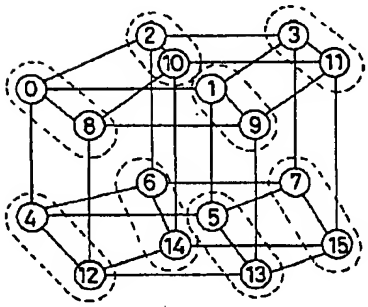
【図１５】



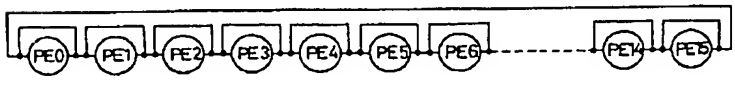
【図７】



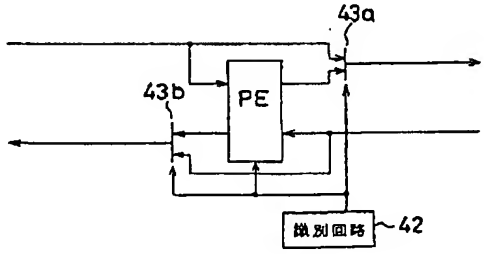
【図１６】



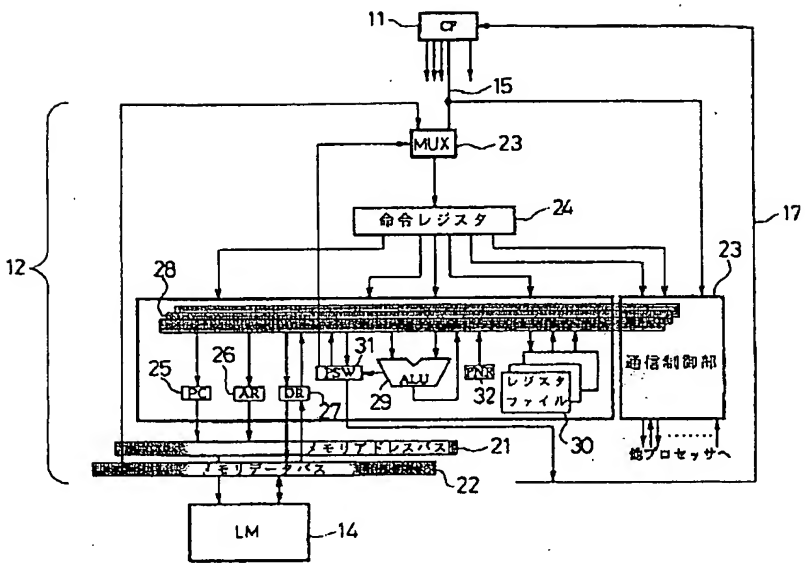
【図１０】



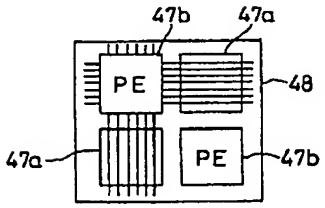
【図１２】



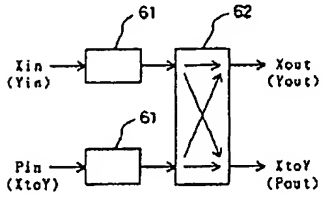
【図９】



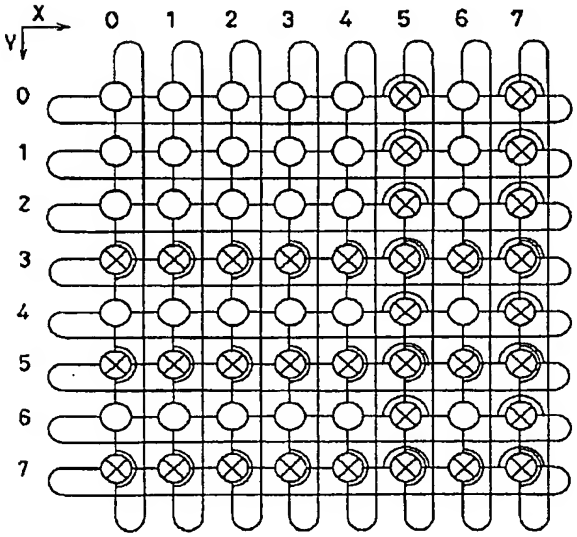
【図１７】



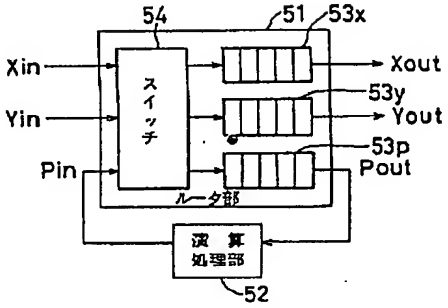
【図２３】



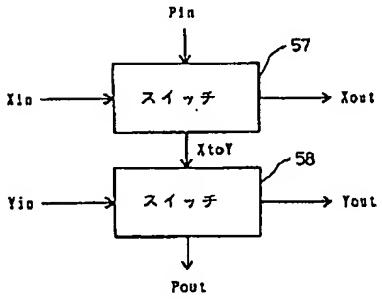
【図１３】



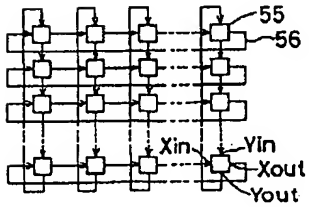
【図１８】



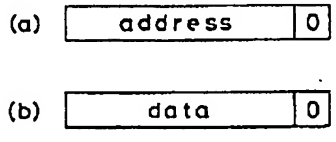
【図２２】



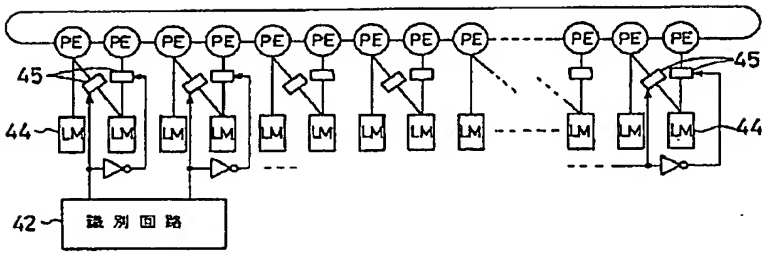
【図１９】



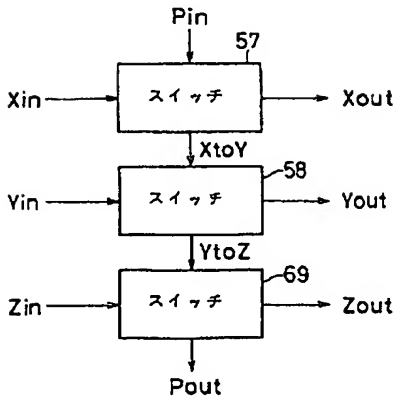
【図２０】



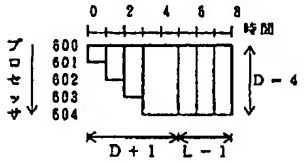
【図１４】



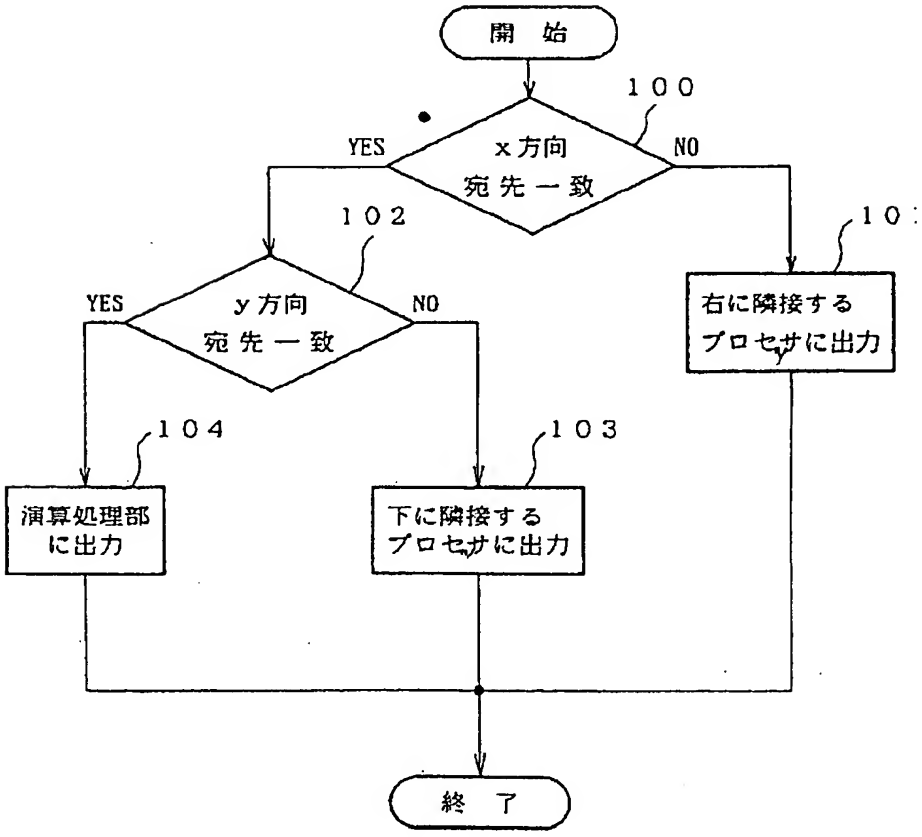
【図２７】



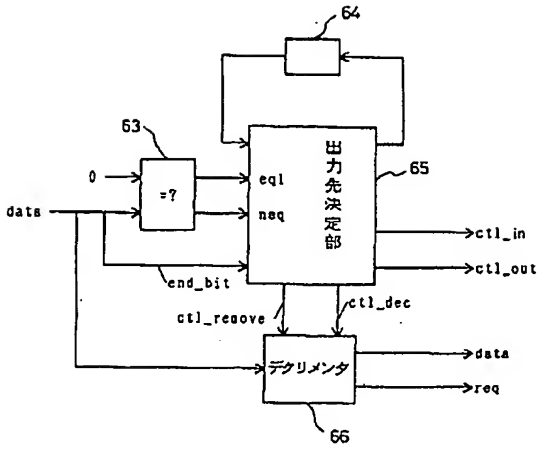
【図３５】



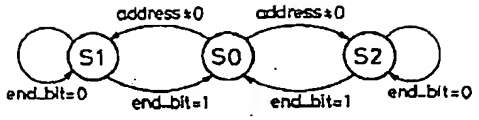
【図２１】



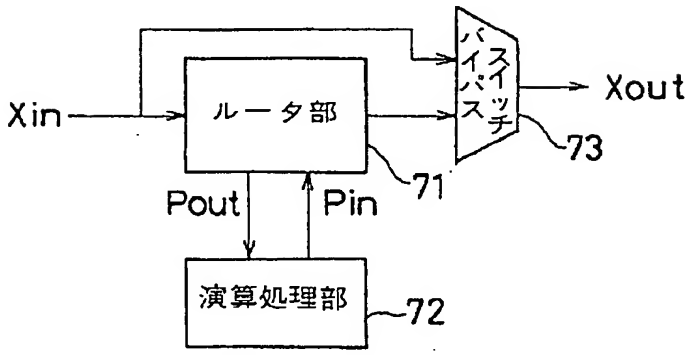
【図２４】



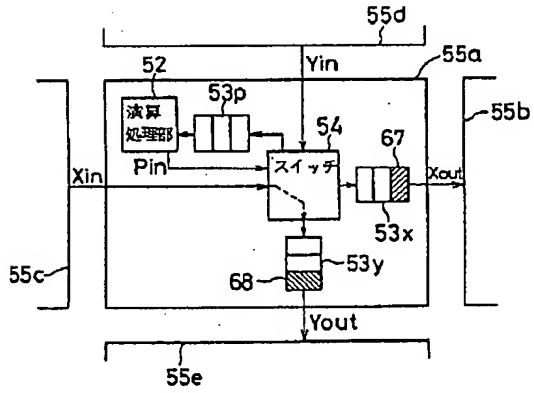
【図２５】



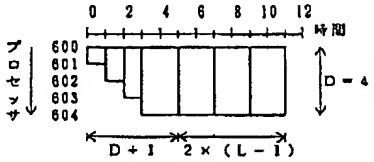
【図２８】



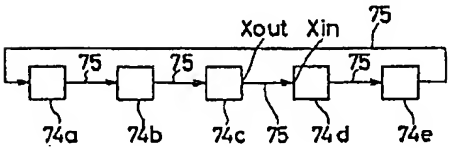
【図２６】



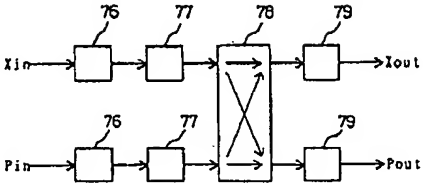
【図３４】



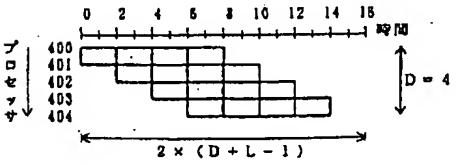
【図２９】



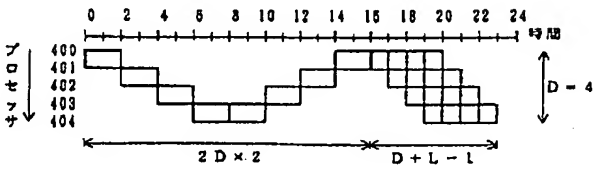
【図３０】



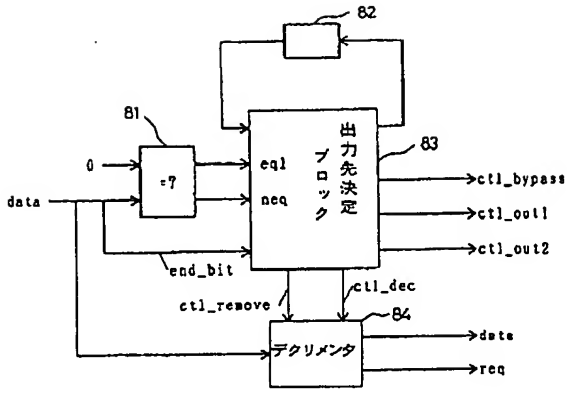
【図４１】



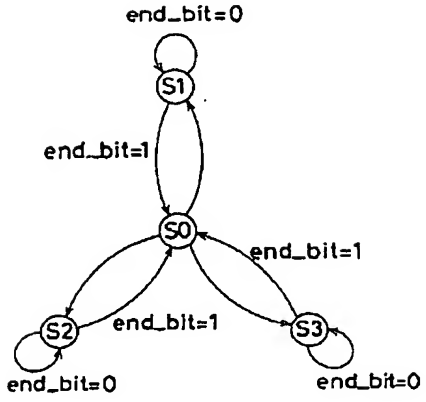
【図４２】



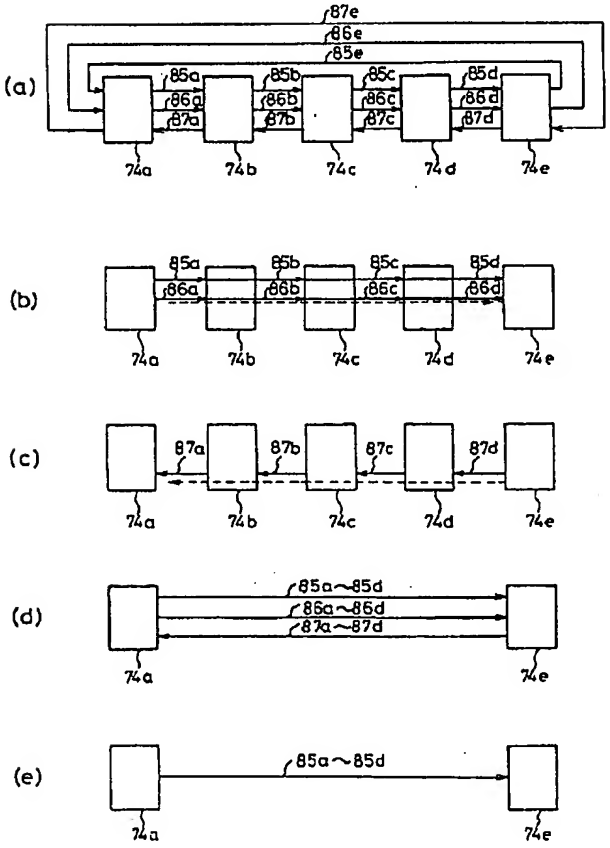
【図３１】



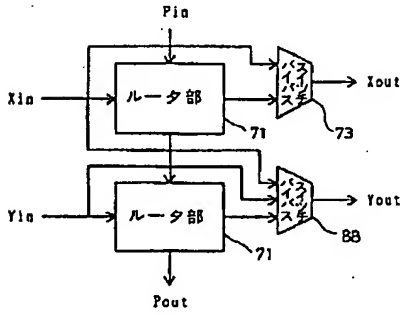
【図３２】



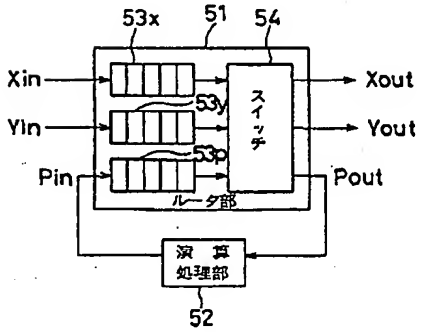
【図３３】



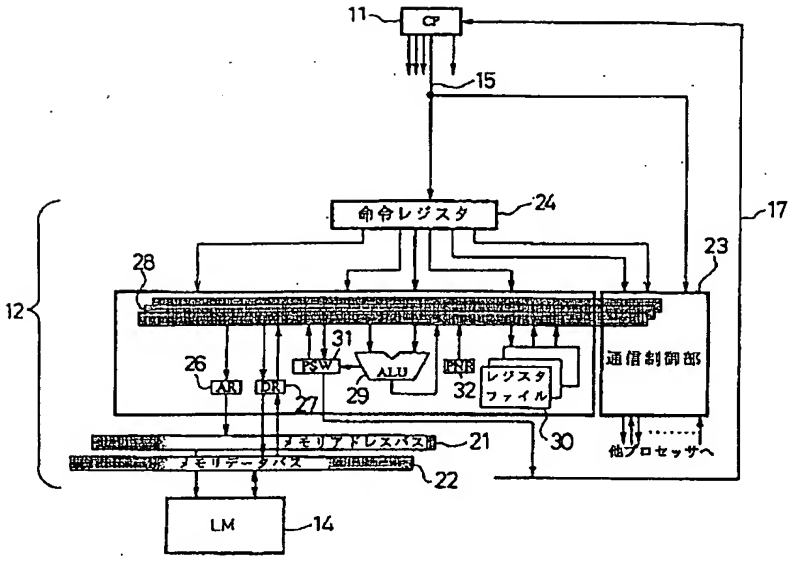
【図３６】



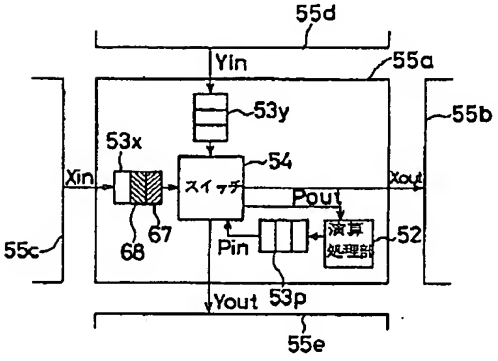
【図３８】



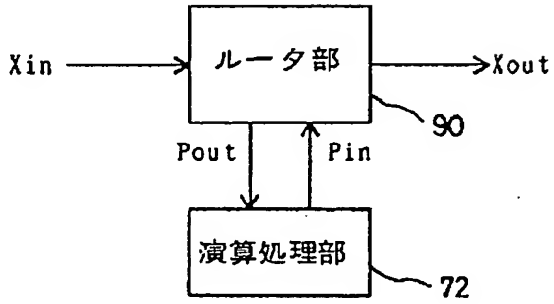
【図３７】



【図３９】



【図４０】



フロントページの続き

(72)発明者　高橋　真史
神奈川県川崎市幸区小向東芝町１　株式会
社東芝総合研究所内